MANUAL TKINTER

LUIS MINGUILLÓN PASCUAL

TKINTER

manual

Luis Minguillón Pascual

ÍNDICE

📘 Manual Completo de Widgets en Tkinter (Python)

1. Introducción a Tkinter y Widgets
2. Tkinter: módulo de interfaces gráficas de Python
3. Widget: cualquier componente visual (botón, etiqueta, cuadro de entrada, etc.)
4. Importación: `import tkinter as tk`, `from tkinter import ttk`
2. Ventana Principal (Tk)
3. Raíz de toda aplicación gráfica
4. Ejemplo de creación y configuración (`root = tk.Tk()`, `root.title()`, `root.geometry()`, `root.mainloop()`)
3. Label (Etiqueta)
4. Muestra texto estático o imágenes
5. Propiedades: `text`, `fg`, `bg`, `font`, `image`
4. Botón (Button)
5. Ejecuta una función al hacer clic
5. Entrada de texto (Entry)
6. Permite al usuario introducir texto de una sola línea
7. Obtener contenido: `.get()`
6. Cuadro de texto (Text)
7. Caja de texto multilínea
8. Obtener contenido: `.get("1.0", tk.END)`
7. Casilla de verificación (Checkbutton)
8. Asociado a una variable (`tk.IntVar()`)
8. Botón de radio (Radiobutton)
9. Útil para selecciones excluyentes
10. Asociado a una variable (`tk.StringVar()`)
9. Botón de comando (Button)
10. Mismas funcionalidades que el Botón (4) con más opciones de estilo (`bg`, `fg`, `padx`, `pady`)
10. Lista desplegable (OptionMenu)
11. Asociado a una variable (`tk.StringVar()`) para la opción seleccionada
11. Lista de selección (Listbox)
12. Permite selección de elementos
13. Modo de selección: `tk.SINGLE`
14. Añadir elementos: `.insert()`
12. Barra de desplazamiento (Scrollbar)
13. Se combina con `Text`, `Canvas`, o `Listbox`
14. Configuración con `yscrollcommand` y `scroll.config(command=texto.yview)`
13. Marco contenedor (Frame)
14. Organiza widgets en secciones
14. Selector de archivos (filedialog)

15.Importación: `from tkinter import filedialog`
16.Función: `filedialog.askopenfilename()`
15.Ventana emergente (Toplevel)
16.Crea una nueva ventana secundaria (`tk.Toplevel(root)`)
16.Spinbox (Spinbox)
17.Selector numérico con rangos (`from_`, `to`)
17.Barra de progreso (Progressbar)
18.Importación: `from tkinter.ttk import Progressbar`
19.Propiedades: `orient`, `length`, `mode`
20.Establecer valor: `pb['value'] = 50`
18.Selector de color (colorchooser)
19.Importación: `from tkinter import colorchooser`
20.Función: `colorchooser.askcolor()`
19.Calendario (Calendar, de tkcalendar)
20.Instalación: `pip install tkcalendar`
21.Importación: `from tkcalendar import Calendar`
20.Administradores de geometría (pack, grid, place)
21.`pack()`: Ordena por bloques (`side`, `fill`, `expand`)
22.`grid()`: Sistema de filas y columnas (`row`, `column`, `padx`, `pady`)
23.`place()`: Posicionamiento absoluto (`x`, `y`)
21.Manejo de eventos y comandos
22.Vinculación de eventos (`.bind("<Button-1>", click)`)
22.Estilización con ttk
23.Widgets modernos (ej. `ttk.Button`, `ttk.Entry`)
24.Importación: `from tkinter import ttk`
23.Ejemplo de aplicación completa
24.Demostración de un formulario simple
24.Recursos adicionales
25.Documentación oficial Tkinter, Guía de ttk, Curso de interfaz gráfica

🧭 **Manual Completo de Clases en Tkinter**

1. Introducción a Tkinter y POO
2. Tkinter como biblioteca estándar de Python
3. POO para encapsular lógica, reutilizar y mantener el código
2. ¿Por qué usar clases en Tkinter?
3. Encapsulación de componentes, separación lógica/presentación, reutilización de widgets, mejor manejo de ventanas
3. Estructura básica de una clase en Tkinter
4. Método `__init__` para inicializar la aplicación y widgets
4. Herencia de tk.Tk y tk.Frame
5. Herencia de `tk.Tk` para la ventana principal
6. Herencia de `tk.Frame` para paneles contenedores
5. Gestión de eventos con métodos
6. Definir funciones como métodos de la clase

6. Separación de componentes en clases
7. Mejora la organización dividiendo la UI en clases individuales
7. Clases y controladores
8. Uso de un controlador para manejar múltiples vistas
8. Ventanas múltiples con clases
9. Creación de ventanas secundarias con `tk.Toplevel` dentro de clases
9. Modularización con múltiples archivos
10. Separación del código en `main.py` e `interfaz.py`
10. Buenas prácticas
11. Cada clase representa un componente, herencia para reutilización, consistencia en administradores de geometría, no mezclar lógica de negocio con interfaz, documentación
11. Ejercicios
12. Crear `FormularioLogin`, `VentanaPrincipal`, `Contador`, clase con `Listbox`, calculadora simple
12. Soluciones (resumen)
13. Ejemplos de implementación para los ejercicios propuestos

📘 **Manual Completo de Opciones en Tkinter**

1. Introducción a las opciones en Tkinter
2. Las opciones determinan apariencia y comportamiento de widgets
2. Sintaxis general de configuración
3. Al crear el widget o con el método `.config()`
3. Opciones comunes a todos los widgets
4. `text`, `bg`/`background`, `fg`/`foreground`, `font`, `padx`/`pady`, `width`/`height`, `state`, `relief`, `anchor`, `cursor`
4. Opciones específicas por widget
5. **Label:** `image`, `justify`, `wraplength`
6. **Button:** `command`, `activebackground`/`activeforeground`, `state`
7. **Entry:** `show`, `textvariable`, `justify`
8. **Text:** `wrap`, `undo`
9. **Checkbutton y Radiobutton:** `variable`, `onvalue`, `offvalue`, `value`
10. **Toplevel y Frame:** `bg`, `bd`, `relief`
5. Métodos config() y cget()
6. `.config()`: cambia opciones en tiempo real
7. `.cget()`: obtiene el valor de una opción
6. Estilos con ttk.Style
7. Para widgets `ttk` (`ttk.Style()`, `style.configure()`)
8. Temas disponibles (`style.theme_names()`, `style.theme_use()`)
7. Colores en Tkinter
8. Nombres estándar ("red", "blue"), códigos HEX ("#FF0000"), RGB (menos común)
8. Fuentes y tipos de letra
9. Formato: `("Nombre", tamaño, "estilo")`
10. Estilos: `bold`, `italic`, `underline`, `overstrike`

9. Gestión del tamaño y el espaciado
10. `padx`, `pady` (espacio interior); `ipadx`, `ipady` (relleno interno); `width`, `height` (en caracteres o líneas)
10. Estados de los widgets
11. `normal`, `disabled`, `active`
11. Ejemplo práctico completo
12. Aplicación para cambiar estilo de etiqueta
12. Recursos adicionales
13. Documentación oficial, guía de opciones (Effbot), listado completo

🎨 **Manual Completo y Extenso sobre Colores en Tkinter**

1. Introducción a los colores en Tkinter
2. Importancia: resaltar, estética, diferenciar, feedback
2. Formatos válidos de colores
3. Nombres de color ("red"), hexadecimal ("#RRGGBB"), escala de grises, colores RGB (conversión externa)
3. Propiedades relacionadas con colores
4. `bg`/`background`, `fg`/`foreground`, `activebackground`, `activeforeground`, `highlightbackground`, `selectbackground`, `disabledforeground`
4. Aplicación de colores a widgets comunes
5. Ejemplos con `Label`, `Button`, `Entry`
5. Paleta básica de colores
6. Nombres comunes y sus códigos HEX (red, green, blue, yellow, cyan, magenta, black, white, gray, orange)
7. Lista completa en `tcl.tk`
6. Uso de colores en temas oscuros y claros
7. Ejemplos de códigos HEX para fondos, textos y resaltados
7. Dinamismo: cambiar colores con eventos
8. Uso de `.bind()` para `FocusIn` y `FocusOut`
8. Personalización con ttk.Style (para widgets ttk)
9. Configuración de estilos (`style.configure("Boton.TButton", ...)`)
10. Limitaciones por sistema operativo
9. Utilidades para generar colores dinámicamente
10. Función para color según estado (`color_estado`)
11. Generación de colores aleatorios (`color_random`)
10. Ejemplos completos
11. Formulario con validación por color
11. Buenas prácticas
12. Nombres hexadecimales, paleta definida, evitar bajo contraste, usar `ttk.Style`, funciones para eventos
12. Ejercicios
13. Crear botones de colores, cambiar color al pasar ratón, formulario de validación por color, `Listbox` con ítems de color, botón personalizado con `ttk.Style`
13. Soluciones (resumen)

14. Ejemplos de código para los ejercicios propuestos

Variables y Operadores

- Programa 1: Sumar dos números
- Programa 2: Restar dos números
- Programa 3: Multiplicar dos números
- Programa 4: Dividir dos números (con control de división por cero)
- Programa 5: Calcular el cuadrado de un número
- Programa 6: Calcular módulo de dos números
- Programa 7: Concatenar dos cadenas
- Programa 8: Verificar si un número es par
- Programa 9: Cambiar signo a un número
- Programa 10: Intercambiar dos valores
- Programa 11: Comparar dos números (mayor o iguales)
- Programa 12: Convertir grados Celsius a Fahrenheit
- Programa 13: Convertir kilómetros a millas
- Programa 14: Calcular promedio de tres números
- Programa 15: Calcular el área de un triángulo
- Programa 16: Verificar si número es positivo, negativo o cero
- Programa 17: Calcular potencia de un número
- Programa 18: Sumar dos números con valores iniciales
- Programa 19: Multiplicación y suma combinadas
- Programa 20: Mostrar número elevado al cubo

Estructuras Condicionales

1. Verificar si un número es positivo o negativo
2. Verificar si un número es par o impar
3. Determinar si una persona es mayor de edad
4. Comparar dos números
5. Evaluar si un año es bisiesto
6. Verificar si un carácter es vocal o consonante
7. Verificar si una contraseña es válida (mínimo 8 caracteres)
8. Clasificar nota en A, B, C, D o F
9. Calcular descuento según edad
10. Determinar si una temperatura está por debajo, en o sobre cero
11. Verificar si un número es múltiplo de 3 y 5
12. Determinar si un número está entre 10 y 100
13. Clasificar un triángulo por lados (Equilátero, Isósceles, Escaleno)
14. Verificar si hay suficientes fondos para una compra
15. Evaluar si un texto contiene una palabra clave ("python")

- Programa 16: Verificar si un año es bisiesto (repetido, misma lógica que el 5)
- Programa 17: Verificar si una palabra es un palíndromo
- Programa 18: Clasificar un número como Bajo, Medio o Alto
- Programa 19: Verificar formato correcto de un nombre (inicial en mayúscula)
- Programa 20: Clasificación por edad (Niño, Adolescente, Adulto, Adulto mayor)

Estructuras Repetitivas. Bucles

- Ejercicio 1: Mostrar los números del 1 al 10 en una etiqueta
- Ejercicio 2: Mostrar los números pares del 2 al 20
- Ejercicio 3: Contar hacia atrás del 10 al 1
- Ejercicio 4: Mostrar 10 veces "Hola"
- Ejercicio 5: Mostrar la suma de los números del 1 al 100
- Ejercicio 6: Contar cuántos múltiplos de 3 hay entre 1 y 100
- Ejercicio 7: Mostrar una tabla de multiplicar del 5
- Ejercicio 8: Repetir una contraseña ingresada 3 veces
- Ejercicio 9: Mostrar los cuadrados del 1 al 10
- Ejercicio 10: Mostrar una cuenta regresiva con botón
- Ejercicio 11: Sumar los dígitos del 12345
- Ejercicio 12: Mostrar el factorial de un número ingresado
- Ejercicio 13: Mostrar números impares del 1 al 19
- Ejercicio 14: Crear una lista con nombres repetidos 5 veces
- Ejercicio 15: Generar la serie Fibonacci hasta el 10º término
- Ejercicio 16: Mostrar 10 líneas con asteriscos crecientes
- Ejercicio 17: Mostrar números del 100 al 50 de 5 en 5
- Ejercicio 18: Sumar los múltiplos de 4 del 1 al 100
- Ejercicio 19: Mostrar la cantidad de letras en cada palabra ingresada
- Ejercicio 20: Repetir un mensaje X veces, ingresando un número

Estructuras de Datos. Listas

- Programa 1: Mostrar los elementos de una lista en una etiqueta
- Programa 2: Agregar elementos a una lista desde una entrada
- Programa 3: Eliminar un elemento de la lista
- Programa 4: Buscar un elemento en la lista
- Programa 5: Contar los elementos de la lista
- Programa 6: Mostrar el elemento más largo de la lista
- Programa 7: Limpiar toda la lista
- Programa 8: Ordenar alfabéticamente una lista
- Programa 9: Mostrar lista invertida
- Programa 10: Mostrar los elementos únicos de una lista
- Programa 11: Mostrar la suma de una lista de números
- Programa 12: Mostrar el valor máximo de la lista
- Programa 13: Mostrar el valor mínimo de la lista
- Programa 14: Insertar un elemento en una posición dada
- Programa 15: Contar ocurrencias de un valor en la lista
- Programa 16: Combinar dos listas y mostrar el resultado
- Programa 17: Reemplazar un valor por otro
- Programa 18: Mostrar solo los valores mayores que 5
- Programa 19: Duplicar todos los elementos numéricos de una lista
- Programa 20: Convertir una lista de números a texto

Estructuras de Datos. Conjuntos

- Ejercicio 1: Crear un conjunto y mostrarlo
- Ejercicio 2: Agregar elemento a conjunto
- Ejercicio 3: Eliminar elemento de conjunto (`discard`)
- Ejercicio 4: Comprobar si un elemento está en el conjunto
- Ejercicio 5: Unión de dos conjuntos
- Ejercicio 6: Intersección de dos conjuntos
- Ejercicio 7: Diferencia entre dos conjuntos (A - B)
- Ejercicio 8: Diferencia simétrica
- Ejercicio 9: Vaciar conjunto (`clear()`)
- Ejercicio 10: Mostrar tamaño del conjunto (`len()`)
- Ejercicio 11: Convertir cadena a conjunto de caracteres
- Ejercicio 12: Crear conjunto desde números separados por coma
- Ejercicio 13: Copiar conjunto (`.copy()`)
- Ejercicio 14: Subconjunto (`issubset()`)
- Ejercicio 15: Superconjunto (`issuperset()`)
- Ejercicio 16: Ver elementos comunes entre dos entradas
- Ejercicio 17: Contar elementos únicos
- Ejercicio 18: Mostrar conjunto ordenado (convertido a lista)
- Ejercicio 19: Encontrar el mayor del conjunto (`max()`)
- Ejercicio 20: Encontrar el menor del conjunto (`min()`)

Estructuras de Datos. Tuplas

- Ejercicio 1: Mostrar una tupla de colores en una etiqueta
- Ejercicio 2: Mostrar el primer elemento de una tupla
- Ejercicio 3: Mostrar los elementos de una tupla en etiquetas separadas
- Ejercicio 4: Mostrar el número de elementos de una tupla
- Ejercicio 5: Buscar un valor en una tupla e indicar si está presente
- Ejercicio 6: Mostrar la suma de los números en una tupla
- Ejercicio 7: Concatenar dos tuplas
- Ejercicio 8: Contar cuántas veces aparece un elemento en una tupla
- Ejercicio 9: Encontrar la posición de un valor en una tupla (`.index()`)
- Ejercicio 10: Convertir una tupla a cadena y mostrarla (`.join()`)
- Ejercicio 11: Mostrar el último elemento de una tupla
- Ejercicio 12: Verificar si la tupla está vacía
- Ejercicio 13: Invertir el orden de una tupla
- Ejercicio 14: Mostrar los elementos de una tupla con separación de líneas
- Ejercicio 15: Convertir lista a tupla y mostrar
- Ejercicio 16: Mostrar una tupla de tuplas
- Ejercicio 17: Mostrar los valores de una tupla de pares clave-valor
- Ejercicio 18: Un botón que muestre un valor de la tupla al azar
- Ejercicio 19: Contar cuántos elementos son mayores a 10 en una tupla
- Ejercicio 20: Mostrar si una tupla contiene solo números

Estructuras de Datos. Diccionarios

- Ejercicio 1: Mostrar el valor de una clave ingresada por el usuario

- Ejercicio 2: Agregar un par clave-valor al diccionario
- Ejercicio 3: Eliminar una clave del diccionario (`.pop()`)
- Ejercicio 4: Verificar si una clave está en el diccionario (`in`)
- Ejercicio 5: Mostrar todas las claves del diccionario (`.keys()`)
- Ejercicio 6: Mostrar todos los valores del diccionario (`.values()`)
- Ejercicio 7: Mostrar todos los elementos (items) del diccionario (`.items()`)
- Ejercicio 8: Contar cuántos elementos tiene el diccionario (`len()`)
- Ejercicio 9: Limpiar el diccionario (`.clear()`)
- Ejercicio 10: Actualizar un valor existente en el diccionario
- Ejercicio 11: Crear un diccionario con nombres y edades desde la interfaz
- Ejercicio 12: Mostrar la edad de una persona a partir de su nombre
- Ejercicio 13: Sumar todos los valores numéricos del diccionario
- Ejercicio 14: Obtener el valor máximo del diccionario
- Ejercicio 15: Copiar un diccionario a otro (`.copy()`)
- Ejercicio 16: Crear un diccionario de conteo de palabras
- Ejercicio 17: Filtrar claves que empiecen por una letra
- Ejercicio 18: Convertir el diccionario en texto plano
- Ejercicio 19: Revertir claves y valores del diccionario
- Ejercicio 20: Mostrar claves cuyos valores sean mayores que un número dado

Funciones

1. Mostrar mensaje con función
2. Sumar dos números
3. Contar clics (con variable global)
4. Cambiar color de fondo
5. Concatenar texto
6. Limpiar texto de un `Entry`
7. Verificar si es número par
8. Multiplicación por 10
9. Convertir a mayúsculas (`.upper()`)
10. Calcular cuadrado (`** 2`)
11. Mostrar longitud de texto (`len()`)
12. Convertir a binario (`bin()`)
13. Mostrar fecha actual (`datetime.date.today()`)
14. Mostrar número aleatorio (`random.randint()`)
15. Determinar vocal
16. Calcular factorial (`math.factorial()`)
17. Convertir Celsius a Fahrenheit
18. Mostrar mensaje personalizado
19. Calcular área de círculo (`math.pi * radio ** 2`)
20. Invertir texto (`[::-1]`)

Programación Orientada a Objetos

- Ejercicio 1: Ventana básica con botón (que cierra la ventana)

- Ejercicio 2: Entrada de texto y etiqueta (mostrar texto de `Entry` en `Label`)
- Ejercicio 3: Contador simple (botón que incrementa número en etiqueta)
- Ejercicio 4: Cambiar color de fondo
- Ejercicio 5: Selector de opciones (usando `Radiobutton`)
- Ejercicio 6: Casillas de verificación (mostrar qué casillas están seleccionadas)
- Ejercicio 7: Cambiar texto de botón
- Ejercicio 8: Saludo personalizado
- Ejercicio 9: Suma de dos números (con validación de entrada)
- Ejercicio 10: Cambiar tamaño de fuente (aumentar letra de etiqueta)
- Ejercicio 11: Calculadora simple (suma, resta, multiplicación, división)
- Ejercicio 12: Temporizador simple (`.after()`)
- Ejercicio 13: Lista desplegable (Combobox de `ttk`)
- Ejercicio 14: Entrada con validación numérica (solo acepta números)
- Ejercicio 15: Mostrar imagen (carga y muestra PNG, requiere Pillow)
- Ejercicio 16: Ventana con pestañas (`ttk.Notebook`)
- Ejercicio 17: Cuadro de diálogo simple (`messagebox.showinfo`)
- Ejercicio 18: Mover etiqueta con botones (`.place()`, `winfo_x()`, `winfo_y()`)
- Ejercicio 19: Crear y eliminar etiquetas dinámicamente
- Ejercicio 20: Aplicación calculadora con historial (guarda operaciones en `Listbox`)

Tratamiento de Ficheros CSV

- Formato y estructura
    - Texto plano, líneas como registros, campos separados por comas
    - No jerárquico, ideal para datos tabulares simples
    - Abrir y mostrar contenido CSV en Text widget
    - Guardar contenido del Text widget en CSV
    - Añadir fila a CSV existente
    - Buscar filas que contengan un texto
    - Contar filas de un CSV
    - Mostrar solo una columna específica
    - Convertir CSV a mayúsculas
    - Eliminar una fila por número
    - Contar ocurrencias de un valor en una columna
    - Reemplazar un valor por otro en todo el CSV
    - Ordenar CSV por una columna
    - Exportar CSV filtrando filas por condición
    - Mostrar estadísticas básicas de una columna numérica (media, mediana, desviación estándar)
    - Dividir un CSV en varios archivos por valor de columna
    - Añadir columna con valores calculados (o fijos)
    - Contar ocurrencias de valores en una columna (usando `collections.Counter`)
    - Mostrar contenido CSV en tabla con Tkinter (`ttk.Treeview`)
    - Actualizar valor en fila y columna específica
    - Eliminar filas con columna vacía
    - Combinar dos CSV y mostrar resultado

Tratamiento de Ficheros JSON

- Formato y estructura
    - Texto estructurado basado en objetos y arrays
    - Puede representar datos complejos y jerárquicos, con anidamientos
    - Muy usado para intercambio de datos web
- Ejercicio 1: Crear una interfaz para guardar un diccionario simple en un fichero JSON
- Ejercicio 2: Leer un archivo JSON y mostrar su contenido en un Text
- Ejercicio 3: Crear una GUI para agregar múltiples personas a un JSON (array de objetos)
- Ejercicio 4: Mostrar nombres almacenados en el JSON en una lista (`Listbox`)
- Ejercicio 5: Buscar una persona por nombre en el JSON
- Ejercicio 6: Editar un nombre ya existente en el fichero JSON
- Ejercicio 7: Eliminar un nombre del JSON
- Ejercicio 8: Guardar y leer múltiples campos (nombre, edad) en JSON
- Ejercicio 9: Leer múltiples campos de un JSON y mostrarlos
- Ejercicio 10: Crear archivo JSON vacío desde Tkinter
- Ejercicio 11: Añadir elementos a un array JSON desde la interfaz
- Ejercicio 12: Mostrar el contenido de un JSON de lista línea por línea
- Ejercicio 13: Eliminar todo el contenido de un JSON desde la interfaz (vaciar el JSON)
- Ejercicio 14: Guardar configuración de usuario en JSON
- Ejercicio 15: Cargar configuración de usuario desde JSON
- Ejercicio 16: Crear interfaz para exportar datos a JSON desde múltiples campos
- Ejercicio 17: Verificar si un archivo JSON existe desde la interfaz (`os.path.exists`)
- Ejercicio 18: Crear un historial de cambios en JSON
- Ejercicio 19: Mostrar el historial desde archivo JSON
- Ejercicio 20: Copiar un JSON a otro archivo desde la interfaz (`shutil.copy`)

📘 Manual Completo de Widgets en Tkinter (Python)

📌 Índice

1. Introducción a Tkinter y Widgets
2. Ventana principal (`Tk`)
3. Label (`Label`)
4. Botón (`Button`)
5. Entrada de texto (`Entry`)
6. Cuadro de texto (`Text`)
7. Casilla de verificación (`Checkbutton`)
8. Botón de radio (`Radiobutton`)
9. Botón de comando (`Button`)
10. Lista desplegable (`OptionMenu`)
11. Lista de selección (`Listbox`)
12. Barra de desplazamiento (`Scrollbar`)
13. Marco contenedor (`Frame`)
14. Selector de archivos (`filedialog`)
15. Ventana emergente (`Toplevel`)
16. Spinbox (`Spinbox`)
17. Barra de progreso (`Progressbar`)
18. Selector de color (`colorchooser`)
19. Calendario (`Calendar`, de `tkcalendar`)
20. Administradores de geometría (`pack`, `grid`, `place`)
21. Manejo de eventos y comandos
22. Estilización con `ttk`
23. Ejemplo de aplicación completa
24. Recursos adicionales

---

1. 🧰 Introducción a Tkinter y Widgets

Tkinter es el módulo de interfaces gráficas de Python. Un **widget** es cualquier componente visual: botón, etiqueta, cuadro de entrada, etc.

```
import tkinter as tk
from tkinter import ttk
```

---

2. 🪟 Ventana Principal (`Tk`)

Es la raíz de toda aplicación gráfica.

```
root = tk.Tk()
root.title("Mi Aplicación")
root.geometry("400x300")
root.mainloop()
```

---

3. 🏷️ `Label` – Etiqueta

Muestra texto estático o imágenes.

```
label = tk.Label(root, text="Hola, mundo", font=("Arial", 14), fg="blue")
label.pack()
```

Propiedades:

- `text`
- `fg` (color del texto)
- `bg` (color de fondo)
- `font`
- `image`

---

4. 🔘 `Button` – Botón

Ejecuta una función al hacer clic.

```
def saludo():
    print("¡Hola!")

boton = tk.Button(root, text="Saludar", command=saludo)
boton.pack()
```

---

5. ✍ `Entry` – Entrada de texto de una sola línea

Permite al usuario introducir texto.

```
entrada = tk.Entry(root)
entrada.pack()

texto = entrada.get()  # Para obtener el contenido
```

---

6. 📄 `Text` – Caja de texto multilínea

```
texto = tk.Text(root, height=5, width=40)
texto.pack()

contenido = texto.get("1.0", tk.END)
```

---

7. ☑ `Checkbutton` – Casilla de verificación

```
var = tk.IntVar()
check = tk.Checkbutton(root, text="Aceptar", variable=var)
check.pack()
```

---

8. 🔘 `Radiobutton` – Botón de opción

Útil para selecciones excluyentes.

```
var = tk.StringVar()
r1 = tk.Radiobutton(root, text="Opción A", variable=var, value="A")
r2 = tk.Radiobutton(root, text="Opción B", variable=var, value="B")
r1.pack()
r2.pack()
```

---

9. 🚀 `Button` – Botón de acción

Mismo que el 4, pero puedes agregar más estilos:

```
boton = tk.Button(root, text="Enviar", bg="green", fg="white", padx=10, pady=5)
```

---

10. 🔽 `OptionMenu` – Menú desplegable

```
opcion = tk.StringVar(value="Rojo")
menu = tk.OptionMenu(root, opcion, "Rojo", "Verde", "Azul")
menu.pack()
```

---

11. 📋 `Listbox` – Lista de selección

```
lista = tk.Listbox(root, selectmode=tk.SINGLE)
lista.insert(1, "Elemento 1")
lista.insert(2, "Elemento 2")
lista.pack()
```

---

12. 📜 `Scrollbar` – Barra de desplazamiento

Se combina con `Text`, `Canvas`, o `Listbox`.

```
scroll = tk.Scrollbar(root)
texto = tk.Text(root, yscrollcommand=scroll.set)
scroll.config(command=texto.yview)
scroll.pack(side=tk.RIGHT, fill=tk.Y)
texto.pack(side=tk.LEFT)
```

---

13. 🧱 `Frame` – Contenedor

Organiza widgets en secciones.

```
frame = tk.Frame(root, bg="lightgray")
frame.pack(pady=10)
```

---

14. 📁 `filedialog` – Selector de archivos

```
from tkinter import filedialog

def abrir_archivo():
    archivo = filedialog.askopenfilename()
    print(archivo)

btn = tk.Button(root, text="Abrir archivo", command=abrir_archivo)
btn.pack()
```

---

15. 🪟 `Toplevel` – Ventana emergente

```
def nueva_ventana():
    ventana = tk.Toplevel(root)
    tk.Label(ventana, text="Soy otra ventana").pack()

btn = tk.Button(root, text="Abrir ventana", command=nueva_ventana)
btn.pack()
```

---

16. 🔢 `Spinbox` – Selector numérico

```
spin = tk.Spinbox(root, from_=0, to=10)
spin.pack()
```

---

17. 📊 `Progressbar` – Barra de progreso

```
from tkinter.ttk import Progressbar

pb = Progressbar(root, orient='horizontal', length=200, mode='determinate')
pb['value'] = 50
pb.pack()
```

---

18. 🎨 `colorchooser` – Selector de color

```
from tkinter import colorchooser

def elegir_color():
    color = colorchooser.askcolor()[1]
    print("Color elegido:", color)

tk.Button(root, text="Color", command=elegir_color).pack()
```

---

19. 🗓️ `Calendar` – Calendario (requiere `tkcalendar`)

```
pip install tkcalendar

from tkcalendar import Calendar

cal = Calendar(root, selectmode='day')
cal.pack()
```

---

20. 📐 Administradores de geometría

`pack()` – Sencillo, ordena por bloques

```
widget.pack(side="top", fill="x", expand=True)
```

`grid()` – Sistema de filas y columnas

```
widget.grid(row=0, column=1, padx=5, pady=5)
```

`place()` – Posicionamiento absoluto

```
widget.place(x=50, y=100)
```

---

21. 🧠 Manejo de eventos

Puedes vincular eventos como clics o teclas:

```
def click(event):
    print("Haz hecho clic")

label = tk.Label(root, text="Haz clic aquí")
label.bind("<Button-1>", click)
label.pack()
```

---

22. 🎨 Estilización con `ttk`

Widgets de `ttk` (como `ttk.Button`, `ttk.Entry`) usan estilos modernos.

```
from tkinter import ttk

btn = ttk.Button(root, text="Botón elegante")
btn.pack()
```

---

23. 🧩 Ejemplo de aplicación completa

```
import tkinter as tk
from tkinter import ttk

def mostrar():
    texto = entrada.get()
    label.config(text=f"Hola, {texto}")

root = tk.Tk()
root.title("Ejemplo Completo")

entrada = tk.Entry(root)
entrada.pack()

btn = ttk.Button(root, text="Saludar", command=mostrar)
btn.pack()

label = tk.Label(root, text="")
label.pack()

root.mainloop()
```

---

24. 📚 Recursos adicionales

- Documentación oficial Tkinter
- Guía de `ttk`
- Curso de interfaz gráfica con Tkinter en Python

🧭 Manual Completo de Clases en Tkinter

ÍNDICE

1. Introducción a Tkinter y POO
2. ¿Por qué usar clases en Tkinter?
3. Estructura básica de una clase en Tkinter
4. Herencia de `tk.Tk` y `tk.Frame`
5. Gestión de eventos con métodos
6. Separación de componentes en clases
7. Clases y controladores
8. Ventanas múltiples con clases
9. Modularización con múltiples archivos
10. Buenas prácticas
11. Ejercicios
12. Soluciones

---

1. Introducción a Tkinter y POO

Tkinter es la biblioteca estándar de Python para crear interfaces gráficas. La POO permite encapsular la lógica de interfaz y su comportamiento en clases, haciendo el código más limpio, reutilizable y mantenible.

---

2. ¿Por qué usar clases en Tkinter?

- Encapsulación de componentes
- Separación entre lógica y presentación
- Reutilización de widgets
- Mejor manejo de múltiples ventanas y componentes

---

3. Estructura básica de una clase en Tkinter

```
import tkinter as tk
```

```
class MiAplicacion:
    def __init__(self, root):
        self.root = root
        self.root.title("Mi Aplicación")
        self.etiqueta = tk.Label(root, text="Hola mundo")
        self.etiqueta.pack()

if __name__ == "__main__":
    root = tk.Tk()
    app = MiAplicacion(root)
    root.mainloop()
```

---

4. Herencia de `tk.Tk` y `tk.Frame`

Heredar de `tk.Tk`

```
class App(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("App")
        tk.Label(self, text="Desde clase App").pack()
```

Heredar de `tk.Frame`

```
class Panel(tk.Frame):
    def __init__(self, master):
        super().__init__(master)
        self.pack()
        tk.Button(self, text="Botón en panel").pack()
```

---

5. Gestión de eventos con métodos

```
class App(tk.Tk):
    def __init__(self):
        super().__init__()
        self.boton = tk.Button(self, text="Haz clic", command=self.saludo)
        self.boton.pack()

    def saludo(self):
        print("Hola desde el método")
```

---

6. Separación de componentes en clases

Separar componentes en clases individuales mejora la organización:

```
class Encabezado(tk.Frame):
    def __init__(self, master):
        super().__init__(master)
        tk.Label(self, text="Encabezado").pack()

class App(tk.Tk):
    def __init__(self):
```

```
        super().__init__()
        self.encabezado = Encabezado(self)
        self.encabezado.pack()
```

---

7. Clases y controladores

Para manejar múltiples vistas, se usa un controlador:

```
class Controlador:
    def __init__(self, root):
        self.root = root
        self.vista = VistaPrincipal(self)

class VistaPrincipal(tk.Frame):
    def __init__(self, controlador):
        super().__init__(controlador.root)
        self.controlador = controlador
        self.pack()
        tk.Button(self, text="Salir",
command=self.controlador.root.destroy).pack()
```

---

8. Ventanas múltiples con clases

```
class VentanaSecundaria(tk.Toplevel):
    def __init__(self, master):
        super().__init__(master)
        self.title("Secundaria")
        tk.Label(self, text="Otra ventana").pack()

class App(tk.Tk):
    def __init__(self):
        super().__init__()
        tk.Button(self, text="Abrir", command=self.abrir_secundaria).pack()

    def abrir_secundaria(self):
        VentanaSecundaria(self)
```

---

9. Modularización con múltiples archivos

main.py

```
from interfaz import App
app = App()
app.mainloop()
```

interfaz.py

```
import tkinter as tk

class App(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("App modular")
```

---

10. Buenas prácticas

- Cada clase debe representar un componente
- Usa herencia para reutilizar código
- Usa `pack`, `grid`, o `place` de forma coherente
- No mezcles lógica de negocio con la interfaz
- Documenta con `docstrings` cada clase

---

11. Ejercicios

1. Crear una clase `FormularioLogin` que tenga campos de usuario y contraseña.
2. Hacer una clase `VentanaPrincipal` que contenga un botón para abrir otra ventana.
3. Crear un `Contador` con un botón de aumentar y otro de reiniciar, todo en clase.
4. Crear una clase que tenga un `Listbox` y botones para añadir/eliminar ítems.
5. Crear una calculadora simple en una clase con herencia de `tk.Tk`.

---

12. Soluciones (resumen)

1. Formulario de Login

```
class FormularioLogin(tk.Frame):
    def __init__(self, master):
        super().__init__(master)
        tk.Label(self, text="Usuario:").pack()
        self.usuario = tk.Entry(self)
        self.usuario.pack()
        tk.Label(self, text="Contraseña:").pack()
        self.clave = tk.Entry(self, show="*")
        self.clave.pack()
```

2. Ventana principal con botón

```
class VentanaPrincipal(tk.Tk):
    def __init__(self):
        super().__init__()
        tk.Button(self, text="Abrir", command=self.abrir).pack()

    def abrir(self):
        tk.Toplevel(self)
```

3. Contador

```
class Contador(tk.Tk):
```

```
    def __init__(self):
        super().__init__()
        self.valor = 0
        self.lbl = tk.Label(self, text=str(self.valor))
        self.lbl.pack()
        tk.Button(self, text="Sumar", command=self.sumar).pack()
        tk.Button(self, text="Reiniciar", command=self.reiniciar).pack()

    def sumar(self):
        self.valor += 1
        self.lbl.config(text=str(self.valor))

    def reiniciar(self):
        self.valor = 0
        self.lbl.config(text="0")
```

📘 Manual Completo de Opciones en Tkinter

📌 Índice

1. Introducción a las opciones en Tkinter
2. Sintaxis general de configuración
3. Opciones comunes a todos los widgets
4. Opciones específicas por widget
5. Métodos `config()` y `cget()`
6. Estilos con `ttk.Style`
7. Colores en Tkinter
8. Fuentes y tipos de letra
9. Gestión del tamaño y el espaciado
10. Estados de los widgets
11. Ejemplo práctico completo
12. Recursos adicionales

---

1. 🧰 Introducción a las opciones en Tkinter

En Tkinter, cada **widget** tiene un conjunto de **opciones** que determinan su apariencia y comportamiento.

Por ejemplo:

```
label = tk.Label(root, text="Hola", bg="blue", fg="white", font=("Arial", 14))
```

---

2. 📝 Sintaxis general de configuración

Puedes establecer opciones de dos maneras:

A) Al crear el widget:

```
boton = tk.Button(root, text="Enviar", bg="green", fg="white")
```

B) Con el método `.config()`:

```
boton.config(text="Confirmar", state="disabled")
```

---

3. 🛠️ Opciones comunes a todos los widgets

Estas opciones son válidas en la mayoría de widgets de Tkinter:

Opción	Descripción
`text`	Texto mostrado en el widget
`bg` / `background`	Color de fondo
`fg` / `foreground`	Color del texto
`font`	Fuente del texto
`padx`, `pady`	Relleno interno en X e Y
`width`, `height`	Ancho y alto del widget
`state`	Estado del widget (`normal`, `disabled`, `active`)
`relief`	Estilo de borde (`flat`, `raised`, `sunken`, `groove`, `ridge`)
`anchor`	Alineación del contenido (`n`, `s`, `e`, `w`, etc.)
`cursor`	Tipo de puntero al pasar el ratón

---

4. 🧩 Opciones específicas por widget

📍 `Label` (etiqueta)

Opción	Descripción
`text`	Texto a mostrar
`image`	Imagen asociada
`justify`	Alineación de texto múltiple
`wraplength`	Longitud máxima de línea

⭕ `Button`

Opción	Descripción
`command`	Función a ejecutar al pulsar
`activebackground`, `activeforeground`	Colores al estar activo
`state`	Estado (`normal`, `disabled`)

✍️ `Entry`

Opción	Descripción
`show`	Sustituto de caracteres (`*`)
`textvariable`	Variable `StringVar`
`justify`	Justificación (`left`, `center`, `right`)

📄 `Text`

Opción	Descripción
`wrap`	Cómo envolver el texto (`none`, `char`, `word`)
`undo`	Permitir deshacer cambios

☑ `Checkbutton` y ⭕ `Radiobutton`

Opción	Descripción
`variable`	Variable asociada (`IntVar`, `StringVar`)
`onvalue`	Valor cuando está activado
`offvalue`	Valor cuando está desactivado
`value`	Valor de selección (`Radiobutton`)

📋 `Listbox`

Opción	Descripción
`selectmode`	Modo de selección (`single`, `multiple`)
`height`	Número de líneas visibles

🪟 `Toplevel` y `Frame`

Opción	Descripción
`bg`	Color de fondo
`bd`	Tamaño del borde
`relief`	Estilo del borde

---

5. 🔧 Métodos `config()` y `cget()`

`.config()` cambia opciones en tiempo real

```
label.config(text="Nuevo texto", fg="red")
```

`.cget()` obtiene el valor de una opción

```
color = label.cget("fg")
print("Color del texto:", color)
```

---

6. 🎨 Estilos con `ttk.Style`

Los widgets `ttk` usan el sistema de estilos:

```
from tkinter import ttk
```

```
style = ttk.Style()
style.configure("TButton", font=("Helvetica", 12), foreground="blue")
```

Temas disponibles:

```
print(style.theme_names())
style.theme_use("clam")  # Cambia el tema
```

---

7. 🎨 Colores en Tkinter

Puedes usar:

- Nombres estándar: `"red"`, `"blue"`, `"green"`
- Códigos HEX: `"#FF0000"`
- RGB con `winfo_rgb()` (menos común)

```
label = tk.Label(root, text="Color", bg="#00FFAA")
```

---

8. 🔤 Fuentes y tipos de letra

```
font = ("Arial", 12, "bold italic underline")
label = tk.Label(root, text="Texto", font=font)
```

- Nombre
- Tamaño
- Estilo: `bold`, `italic`, `underline`, `overstrike`

---

9. 📏 Tamaño y espaciado

- `padx`, `pady`: espacio interior
- `ipadx`, `ipady`: relleno interno del contenido
- `width`, `height`: en caracteres o líneas

```
label = tk.Label(root, text="Ancho", width=20, height=2, padx=5, pady=10)
```

---

10. 🚦 Estados de los widgets

Estado	Descripción
`normal`	Activo, se puede usar
`disabled`	Deshabilitado, no se puede usar
`active`	Solo visible en botones al pasar el ratón

```
btn.config(state="disabled")
```

---

11. 🧩 Ejemplo práctico completo

```
import tkinter as tk

def cambiar_color():
    etiqueta.config(fg="green", font=("Verdana", 16, "bold"))

root = tk.Tk()
root.title("Opciones en Tkinter")

etiqueta = tk.Label(root, text="Texto original", fg="black", font=("Arial", 12))
etiqueta.pack(pady=10)

boton = tk.Button(root, text="Cambiar estilo", command=cambiar_color)
boton.pack(pady=10)

root.mainloop()
```

---

12. 📚 Recursos adicionales

- Documentación oficial de Tkinter
- Guía de opciones de widgets en Tkinter (Effbot)
- Listado completo de opciones

🎨 Manual Completo y Extenso sobre Colores en Tkinter

ÍNDICE

1. Introducción a los colores en Tkinter
2. Formatos válidos de colores
3. Propiedades relacionadas con colores
4. Aplicación de colores a widgets comunes
5. Paleta básica de colores
6. Uso de colores en temas oscuros y claros
7. Dinamismo: cambiar colores con eventos
8. Personalización con `ttk.Style`
9. Utilidades para generar colores dinámicamente
10. Ejemplos completos
11. Buenas prácticas
12. Ejercicios
13. Soluciones

---

1. Introducción a los colores en Tkinter

Tkinter permite aplicar colores a casi todos los widgets. Los colores son importantes para:

- Resaltar información
- Mejorar la estética
- Diferenciar secciones
- Dar feedback al usuario

---

2. Formatos válidos de colores

Tkinter acepta los siguientes formatos de color:

- **Nombre de color:** `"red"`, `"blue"`, `"green"`
- **Hexadecimal:** `"#RRGGBB"` (ej. `#ff0000`)

- **Escala de grises:** `"#AAAAAA"` (misma intensidad en R, G, B)
- **Colores personalizados RGB (solo en algunas versiones):**
  No directamente soportado por nombres RGB, pero puedes convertir con bibliotecas externas.

---

3. Propiedades relacionadas con colores

Estas son las propiedades comunes de color para widgets:

Propiedad	Descripción
`bg` o `background`	Color de fondo
`fg` o `foreground`	Color del texto
`activebackground`	Fondo cuando el mouse está encima
`activeforeground`	Texto cuando el mouse está encima
`highlightbackground`	Borde enfocado
`selectbackground`	Fondo del texto seleccionado
`disabledforeground`	Texto cuando está deshabilitado

---

4. Aplicación de colores a widgets comunes

Ejemplo con `Label`:

```
tk.Label(root, text="Texto rojo", fg="red", bg="black")
```

Ejemplo con `Button`:

```
tk.Button(root, text="Botón", bg="lightblue", activebackground="blue")
```

Ejemplo con `Entry`:

```
tk.Entry(root, bg="#ffffcc", fg="black", insertbackground="blue")
```

---

5. Paleta básica de colores

Tkinter reconoce más de 140 nombres de color.

Nombre	Hex	Resultado
red	#FF0000	
green	#00FF00	
blue	#0000FF	
yellow	#FFFF00	
cyan	#00FFFF	
magenta	#FF00FF	

Nombre	Hex	Resultado
black	#000000	●
white	#FFFFFF	○
gray	#808080	☐
orange	#FFA500	🟠

Puedes consultar la lista completa en:
https://www.tcl.tk/man/tcl8.4/TkCmd/colors.htm

---

6. Uso de colores en temas oscuros y claros

Para temas **oscuros**:

```
bg = "#1e1e1e"
fg = "#ffffff"
highlight = "#3c3c3c"
```

Para temas **claros**:

```
bg = "#f0f0f0"
fg = "#000000"
highlight = "#cccccc"
```

---

7. Dinamismo: cambiar colores con eventos

```
def al_entrar(event):
    event.widget.config(bg="lightgreen")

def al_salir(event):
    event.widget.config(bg="white")

entrada = tk.Entry(root)
entrada.pack()
entrada.bind("<FocusIn>", al_entrar)
entrada.bind("<FocusOut>", al_salir)
```

---

8. Personalización con `ttk.Style` (para widgets `ttk`)

```
from tkinter import ttk

style = ttk.Style()
style.configure("Boton.TButton", background="blue", foreground="white")

ttk.Button(root, text="Botón Estilizado", style="Boton.TButton").pack()
```

Ten en cuenta que `ttk` limita algunos estilos por sistema operativo. En Windows y Linux, puedes cambiar más cosas que en macOS.

---

9. Utilidades para generar colores dinámicamente

Cambiar color en función del estado

```
def color_estado(estado):
    return {
        "bien": "#ccffcc",
        "error": "#ffcccc",
        "neutro": "#f0f0f0"
    }.get(estado, "#ffffff")
```

Colores aleatorios

```
import random

def color_random():
    return "#%06x" % random.randint(0, 0xFFFFFF)
```

---

10. Ejemplos completos

Ejemplo 1: Formulario con validación por color

```
import tkinter as tk

class Formulario(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Formulario")

        self.entrada = tk.Entry(self)
        self.entrada.pack()
        self.boton = tk.Button(self, text="Validar", command=self.validar)
        self.boton.pack()

    def validar(self):
        texto = self.entrada.get()
        if texto == "ok":
            self.entrada.config(bg="#ccffcc")
        else:
            self.entrada.config(bg="#ffcccc")

app = Formulario()
app.mainloop()
```

---

11. Buenas prácticas

- Usa nombres hexadecimales para consistencia
- Define una paleta al principio del programa
- Evita combinaciones con bajo contraste (ej. texto amarillo sobre blanco)
- Usa estilos (`ttk.Style`) para widgets reutilizables

- Define funciones para cambiar colores en eventos

---

12. Ejercicios

1. Crea una interfaz con tres botones de colores distintos.
2. Cambia el color de fondo de un widget cuando el ratón entra y sale.
3. Diseña un formulario que valide la longitud del texto con colores.
4. Crea un `Listbox` donde los ítems seleccionados se pinten de un color diferente.
5. Usa `ttk.Style` para crear un botón personalizado azul con texto blanco.

---

13. Soluciones (resumen)

1. Botones de colores

```
tk.Button(root, text="Rojo", bg="red").pack()
tk.Button(root, text="Verde", bg="green").pack()
tk.Button(root, text="Azul", bg="blue").pack()
```

2. Ratón entra y sale

```
def entra(e): e.widget.config(bg="lightgreen")
def sale(e): e.widget.config(bg="white")
entrada = tk.Entry(root)
entrada.pack()
entrada.bind("<Enter>", entra)
entrada.bind("<Leave>", sale)
```

Ejercicios

1. Variables y operadores
2. Estructuras condicionales
3. Estructuras repetitivas. Bucles
4. Estructuras de datos. Listas
5. Estructura de datos. Conjuntos
6. Estructuras de datos. Tuplas
7. Estructuras de datos. Diccionarios
8. Funciones
9. Programación orientada a objetos
10. Tratamiento de ficheros CSV
11. Tratamiento de ficheros JSON

1. VARIABLES Y OPERADORES

Programa 1: Sumar dos números

```
import tkinter as tk

def sumar():
    resultado = int(entrada1.get()) + int(entrada2.get())
    etiqueta_resultado.config(text=f"Resultado: {resultado}")

ventana = tk.Tk()
entrada1 = tk.Entry(ventana)
entrada2 = tk.Entry(ventana)
boton = tk.Button(ventana, text="Sumar", command=sumar)
etiqueta_resultado = tk.Label(ventana, text="Resultado:")

entrada1.pack()
entrada2.pack()
boton.pack()
etiqueta_resultado.pack()
ventana.mainloop()
```

Explicación: El programa suma dos números ingresados y muestra el resultado.

Programa 2: Restar dos números

```
import tkinter as tk

def restar():
    resultado = int(entrada1.get()) - int(entrada2.get())
    etiqueta_resultado.config(text=f"Resultado: {resultado}")

ventana = tk.Tk()
entrada1 = tk.Entry(ventana)
entrada2 = tk.Entry(ventana)
boton = tk.Button(ventana, text="Restar", command=restar)
etiqueta_resultado = tk.Label(ventana, text="Resultado:")

entrada1.pack()
entrada2.pack()
boton.pack()
etiqueta_resultado.pack()
ventana.mainloop()
```

Explicación: Resta los valores ingresados en dos campos y los muestra.

Programa 3: Multiplicar dos números

```
import tkinter as tk

def multiplicar():
    resultado = int(entrada1.get()) * int(entrada2.get())
    etiqueta_resultado.config(text=f"Resultado: {resultado}")

ventana = tk.Tk()
entrada1 = tk.Entry(ventana)
entrada2 = tk.Entry(ventana)
boton = tk.Button(ventana, text="Multiplicar",
command=multiplicar)
etiqueta_resultado = tk.Label(ventana, text="Resultado:")

entrada1.pack()
entrada2.pack()
boton.pack()
etiqueta_resultado.pack()
ventana.mainloop()
```

Explicación: Multiplica dos números y muestra el resultado.

Programa 4: Dividir dos números

```
import tkinter as tk

def dividir():
    try:
        resultado = int(entrada1.get()) / int(entrada2.get())
        etiqueta_resultado.config(text=f"Resultado: {resultado}")
    except ZeroDivisionError:
        etiqueta_resultado.config(text="No se puede dividir entre
cero")

ventana = tk.Tk()
entrada1 = tk.Entry(ventana)
entrada2 = tk.Entry(ventana)
boton = tk.Button(ventana, text="Dividir", command=dividir)
etiqueta_resultado = tk.Label(ventana, text="Resultado:")

entrada1.pack()
entrada2.pack()
boton.pack()
etiqueta_resultado.pack()
ventana.mainloop()
```

Explicación: Realiza una división controlando la división por cero.

Programa 5: Calcular el cuadrado de un número

```
import tkinter as tk

def cuadrado():
    resultado = int(entrada.get()) ** 2
    etiqueta_resultado.config(text=f"Cuadrado: {resultado}")

ventana = tk.Tk()
entrada = tk.Entry(ventana)
boton = tk.Button(ventana, text="Cuadrado", command=cuadrado)
etiqueta_resultado = tk.Label(ventana, text="Cuadrado:")

entrada.pack()
boton.pack()
etiqueta_resultado.pack()
ventana.mainloop()
```

Explicación: Calcula el cuadrado del número ingresado.

Programa 6: Calcular módulo de dos números

```
import tkinter as tk

def modulo():
    resultado = int(entrada1.get()) % int(entrada2.get())
    etiqueta_resultado.config(text=f"Módulo: {resultado}")

ventana = tk.Tk()
entrada1 = tk.Entry(ventana)
entrada2 = tk.Entry(ventana)
boton = tk.Button(ventana, text="Calcular módulo", command=modulo)
etiqueta_resultado = tk.Label(ventana, text="Módulo:")

entrada1.pack()
entrada2.pack()
boton.pack()
etiqueta_resultado.pack()
ventana.mainloop()
```

Explicación: Usa el operador módulo para mostrar el resto de una división.

Programa 7: Concatenar dos cadenas

```
import tkinter as tk

def concatenar():
    resultado = entrada1.get() + " " + entrada2.get()
    etiqueta_resultado.config(text=f"Resultado: {resultado}")
```

```
ventana = tk.Tk()
entrada1 = tk.Entry(ventana)
entrada2 = tk.Entry(ventana)
boton = tk.Button(ventana, text="Concatenar", command=concatenar)
etiqueta_resultado = tk.Label(ventana, text="Resultado:")

entrada1.pack()
entrada2.pack()
boton.pack()
etiqueta_resultado.pack()
ventana.mainloop()
```

Explicación: Une dos textos separados por espacio.

Programa 8: Verificar si un número es par

```
import tkinter as tk

def es_par():
    numero = int(entrada.get())
    resultado = "Par" if numero % 2 == 0 else "Impar"
    etiqueta_resultado.config(text=resultado)

ventana = tk.Tk()
entrada = tk.Entry(ventana)
boton = tk.Button(ventana, text="Verificar", command=es_par)
etiqueta_resultado = tk.Label(ventana, text="")

entrada.pack()
boton.pack()
etiqueta_resultado.pack()
ventana.mainloop()
```

Explicación: Comprueba si el número ingresado es par o impar.

Programa 9: Cambiar signo a un número

```
import tkinter as tk

def cambiar_signo():
    numero = int(entrada.get())
    resultado = -numero
    etiqueta_resultado.config(text=f"Resultado: {resultado}")

ventana = tk.Tk()
entrada = tk.Entry(ventana)
boton = tk.Button(ventana, text="Cambiar signo",
command=cambiar_signo)
etiqueta_resultado = tk.Label(ventana, text="")
```

```
entrada.pack()
boton.pack()
etiqueta_resultado.pack()
ventana.mainloop()
```

Explicación: Cambia el signo del número ingresado.

Programa 10: Intercambiar dos valores

```
import tkinter as tk

def intercambiar():
    a = entrada1.get()
    b = entrada2.get()
    entrada1.delete(0, tk.END)
    entrada2.delete(0, tk.END)
    entrada1.insert(0, b)
    entrada2.insert(0, a)

ventana = tk.Tk()
entrada1 = tk.Entry(ventana)
entrada2 = tk.Entry(ventana)
boton = tk.Button(ventana, text="Intercambiar",
command=intercambiar)

entrada1.pack()
entrada2.pack()
boton.pack()
ventana.mainloop()
```

Explicación: Intercambia los valores entre dos cajas de entrada.

Programa 11: Comparar dos números (mayor)

```
import tkinter as tk


def mayor():
    a = int(entrada1.get())
    b = int(entrada2.get())
    if a > b:
        resultado = f"{a} es mayor"
    elif b > a:
        resultado = f"{b} es mayor"
    else:
```

```
        resultado = "Son iguales"
    etiqueta_resultado.config(text=resultado)

ventana = tk.Tk()
entrada1 = tk.Entry(ventana)
entrada2 = tk.Entry(ventana)
boton = tk.Button(ventana, text="Comparar", command=mayor)
etiqueta_resultado = tk.Label(ventana, text="")

entrada1.pack()
entrada2.pack()
boton.pack()
etiqueta_resultado.pack()
ventana.mainloop()
```

Explicación: Compara dos números e indica cuál es mayor o si son iguales.

Programa 12: Convertir grados Celsius a Fahrenheit

```
import tkinter as tk

def convertir():
    c = float(entrada.get())
    f = (c * 9/5) + 32
    etiqueta_resultado.config(text=f"{f} °F")

ventana = tk.Tk()
entrada = tk.Entry(ventana)
boton = tk.Button(ventana, text="Convertir", command=convertir)
etiqueta_resultado = tk.Label(ventana, text="")

entrada.pack()
boton.pack()
etiqueta_resultado.pack()
```

```
ventana.mainloop()
```

Explicación: Convierte temperatura de grados Celsius a Fahrenheit.

Programa 13: Convertir kilómetros a millas

```
import tkinter as tk

def convertir():
    km = float(entrada.get())
    millas = km * 0.621371
    etiqueta_resultado.config(text=f"{millas} millas")

ventana = tk.Tk()
entrada = tk.Entry(ventana)
boton = tk.Button(ventana, text="Convertir", command=convertir)
etiqueta_resultado = tk.Label(ventana, text="")

entrada.pack()
boton.pack()
etiqueta_resultado.pack()
ventana.mainloop()
```

Explicación: Multiplica kilómetros por 0.621371 para obtener millas.

Programa 14: Calcular promedio de tres números

```
import tkinter as tk

def promedio():
    a = float(entrada1.get())
    b = float(entrada2.get())
    c = float(entrada3.get())
    prom = (a + b + c) / 3
    etiqueta_resultado.config(text=f"Promedio: {prom}")
```

```
ventana = tk.Tk()
entrada1 = tk.Entry(ventana)
entrada2 = tk.Entry(ventana)
entrada3 = tk.Entry(ventana)
boton = tk.Button(ventana, text="Calcular", command=promedio)
etiqueta_resultado = tk.Label(ventana, text="")

entrada1.pack()
entrada2.pack()
entrada3.pack()
boton.pack()
etiqueta_resultado.pack()
ventana.mainloop()
```

Explicación: Calcula el promedio de tres números introducidos.

Programa 15: Calcular el área de un triángulo

```
import tkinter as tk

def calcular_area():
    base = float(entrada1.get())
    altura = float(entrada2.get())
    area = (base * altura) / 2
    etiqueta_resultado.config(text=f"Área: {area}")

ventana = tk.Tk()
entrada1 = tk.Entry(ventana)
entrada2 = tk.Entry(ventana)
boton = tk.Button(ventana, text="Calcular área",
command=calcular_area)
etiqueta_resultado = tk.Label(ventana, text="")

entrada1.pack()
```

```
entrada2.pack()
boton.pack()
etiqueta_resultado.pack()
ventana.mainloop()
```

Explicación: Aplica la fórmula del área de un triángulo (base x altura / 2).

Programa 16: Verificar si número es positivo, negativo o cero

```
import tkinter as tk

def verificar():
    n = int(entrada.get())
    if n > 0:
        mensaje = "Positivo"
    elif n < 0:
        mensaje = "Negativo"
    else:
        mensaje = "Cero"
    etiqueta_resultado.config(text=mensaje)

ventana = tk.Tk()
entrada = tk.Entry(ventana)
boton = tk.Button(ventana, text="Verificar", command=verificar)
etiqueta_resultado = tk.Label(ventana, text="")

entrada.pack()
boton.pack()
etiqueta_resultado.pack()
ventana.mainloop()
```

Explicación: Usa condicionales para mostrar si el número es positivo, negativo o cero.

Programa 17: Calcular potencia de un número

```
import tkinter as tk

def potencia():
    base = int(entrada1.get())
    exponente = int(entrada2.get())
    resultado = base ** exponente
    etiqueta_resultado.config(text=f"Resultado: {resultado}")

ventana = tk.Tk()
entrada1 = tk.Entry(ventana)
entrada2 = tk.Entry(ventana)
boton = tk.Button(ventana, text="Potencia", command=potencia)
etiqueta_resultado = tk.Label(ventana, text="")

entrada1.pack()
entrada2.pack()
boton.pack()
etiqueta_resultado.pack()
ventana.mainloop()
```

Explicación: Calcula la potencia de un número usando `**`.

Programa 18: Sumar dos números con valores iniciales

```
import tkinter as tk

def sumar():
    a = 10
    b = 5
    resultado = a + b
    etiqueta_resultado.config(text=f"10 + 5 = {resultado}")

ventana = tk.Tk()
```

```
boton = tk.Button(ventana, text="Sumar", command=sumar)
etiqueta_resultado = tk.Label(ventana, text="")

boton.pack()
etiqueta_resultado.pack()
ventana.mainloop()
```

Explicación: Usa variables internas sin entrada de usuario.

Programa 19: Multiplicación y suma combinadas

```
import tkinter as tk

def operar():
    a = int(entrada1.get())
    b = int(entrada2.get())
    resultado = (a + b) * 2
    etiqueta_resultado.config(text=f"Resultado: {resultado}")

ventana = tk.Tk()
entrada1 = tk.Entry(ventana)
entrada2 = tk.Entry(ventana)
boton = tk.Button(ventana, text="Operar", command=operar)
etiqueta_resultado = tk.Label(ventana, text="")

entrada1.pack()
entrada2.pack()
boton.pack()
etiqueta_resultado.pack()
ventana.mainloop()
```

Explicación: Suma los valores y luego los multiplica por 2.

Programa 20: Mostrar número elevado al cubo

```
import tkinter as tk

def al_cubo():
    n = int(entrada.get())
    resultado = n ** 3
    etiqueta_resultado.config(text=f"Cubo: {resultado}")

ventana = tk.Tk()
entrada = tk.Entry(ventana)
boton = tk.Button(ventana, text="Calcular cubo", command=al_cubo)
etiqueta_resultado = tk.Label(ventana, text="")

entrada.pack()
boton.pack()
etiqueta_resultado.pack()
ventana.mainloop()
```

Explicación: Calcula y muestra el número elevado al cubo.

---

2. ESTRUCTURAS CONDICIONALES

1. Verificar si un número es positivo o negativo

```
import tkinter as tk

def verificar():
    num = int(entrada.get())
    if num >= 0:
        resultado.config(text="Es positivo")
    else:
        resultado.config(text="Es negativo")

ventana = tk.Tk()
ventana.title("Número positivo o negativo")

tk.Label(ventana, text="Introduce un número:").pack()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Verificar", command=verificar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: El usuario introduce un número y se evalúa si es positivo o negativo mediante una estructura if-else.

---

2. Verificar si un número es par o impar

```
import tkinter as tk

def verificar():
    num = int(entrada.get())
    if num % 2 == 0:
        resultado.config(text="Es par")
    else:
        resultado.config(text="Es impar")

ventana = tk.Tk()
ventana.title("Par o impar")

tk.Label(ventana, text="Número:").pack()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Verificar", command=verificar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Se usa la condición `num % 2 == 0` para verificar si el número es par.

---

3. Determinar si una persona es mayor de edad

```
import tkinter as tk

def verificar():
```

```
    edad = int(entrada.get())
    if edad >= 18:
        resultado.config(text="Mayor de edad")
    else:
        resultado.config(text="Menor de edad")

ventana = tk.Tk()
ventana.title("Mayoría de edad")

tk.Label(ventana, text="Edad:").pack()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Verificar", command=verificar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Se evalúa si la edad ingresada es mayor o igual a 18.

---

4. Comparar dos números

```
import tkinter as tk

def comparar():
    a = int(num1.get())
    b = int(num2.get())
    if a > b:
        resultado.config(text="El primero es mayor")
    elif a < b:
        resultado.config(text="El segundo es mayor")
    else:
        resultado.config(text="Son iguales")

ventana = tk.Tk()
ventana.title("Comparar dos números")

tk.Label(ventana, text="Número 1:").pack()
num1 = tk.Entry(ventana)
num1.pack()
tk.Label(ventana, text="Número 2:").pack()
num2 = tk.Entry(ventana)
num2.pack()
tk.Button(ventana, text="Comparar", command=comparar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Se comparan dos números usando `if`, `elif` y `else`.

---

5. Evaluar si un año es bisiesto

```
import tkinter as tk

def bisiesto():
    año = int(entrada.get())
    if (año % 4 == 0 and año % 100 != 0) or (año % 400 == 0):
        resultado.config(text="Es bisiesto")
```

```
    else:
        resultado.config(text="No es bisiesto")

ventana = tk.Tk()
ventana.title("Año bisiesto")

tk.Label(ventana, text="Año:").pack()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Verificar", command=bisiesto).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Se evalúa la condición para que un año sea bisiesto.

---

6. Verificar si un carácter es vocal o consonante

```
import tkinter as tk


def verificar():
    letra = entrada.get().lower()
    if letra in 'aeiou':
        resultado.config(text="Es vocal")
    else:
        resultado.config(text="Es consonante")


ventana = tk.Tk()
ventana.title("Vocal o consonante")


tk.Label(ventana, text="Letra:").pack()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Verificar", command=verificar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()


ventana.mainloop()
```

Explicación: Se comprueba si la letra ingresada está en el conjunto de vocales.

---

7. Verificar si una contraseña es válida (mínimo 8 caracteres)

```
import tkinter as tk
```

```
def verificar():
    clave = entrada.get()
    if len(clave) >= 8:
        resultado.config(text="Contraseña válida")
    else:
        resultado.config(text="Contraseña inválida")

ventana = tk.Tk()
ventana.title("Verificar contraseña")

tk.Label(ventana, text="Contraseña:").pack()
entrada = tk.Entry(ventana, show="*")
entrada.pack()
tk.Button(ventana, text="Verificar", command=verificar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Se usa `len()` para verificar la longitud de la cadena ingresada.

---

8. Clasificar nota en A, B, C, D o F

```
import tkinter as tk

def clasificar():
    nota = int(entrada.get())
    if nota >= 90:
        resultado.config(text="A")
    elif nota >= 80:
        resultado.config(text="B")
    elif nota >= 70:
        resultado.config(text="C")
    elif nota >= 60:
        resultado.config(text="D")
    else:
        resultado.config(text="F")
```

```
ventana = tk.Tk()
ventana.title("Clasificación de nota")

tk.Label(ventana, text="Nota:").pack()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Clasificar", command=clasificar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Se usa una serie de condiciones anidadas para determinar la calificación.

---

9. Calcular descuento según edad

```
import tkinter as tk

def calcular():
    edad = int(entrada.get())
    if edad < 18:
        resultado.config(text="Descuento: 20%")
    elif edad <= 60:
        resultado.config(text="Descuento: 10%")
    else:
        resultado.config(text="Descuento: 30%")

ventana = tk.Tk()
ventana.title("Descuento por edad")

tk.Label(ventana, text="Edad:").pack()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Calcular", command=calcular).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Se asigna un porcentaje de descuento según el rango de edad.

---

10. Determinar si una temperatura está por debajo, en o sobre cero

```
import tkinter as tk

def temperatura():
    temp = float(entrada.get())
    if temp < 0:
        resultado.config(text="Bajo cero")
    elif temp == 0:
        resultado.config(text="Punto de congelación")
    else:
        resultado.config(text="Sobre cero")

ventana = tk.Tk()
ventana.title("Evaluar temperatura")

tk.Label(ventana, text="Temperatura:").pack()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Evaluar", command=temperatura).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Se evalúa si una temperatura es negativa, cero o positiva.

---

11. Verificar si un número es múltiplo de 3 y 5

```
import tkinter as tk

def verificar():
    num = int(entrada.get())
    if num % 3 == 0 and num % 5 == 0:
        resultado.config(text="Múltiplo de 3 y 5")
    else:
        resultado.config(text="No lo es")
```

```
ventana = tk.Tk()
ventana.title("Múltiplo de 3 y 5")

tk.Label(ventana, text="Número:").pack()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Verificar", command=verificar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Se usa `and` para comprobar que sea múltiplo de ambos.

---

12. Determinar si un número está entre 10 y 100

```
import tkinter as tk

def verificar():
    num = int(entrada.get())
    if 10 <= num <= 100:
        resultado.config(text="Está entre 10 y 100")
    else:
        resultado.config(text="No está en el rango")

ventana = tk.Tk()
ventana.title("Rango de número")

tk.Label(ventana, text="Número:").pack()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Verificar", command=verificar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Se usa el operador de comparación compuesta `10 <= num <= 100`.

---

13. Clasificar un triángulo por lados

```
import tkinter as tk

def clasificar():
    a = int(lado1.get())
    b = int(lado2.get())
    c = int(lado3.get())
    if a == b == c:
        resultado.config(text="Equilátero")
    elif a == b or b == c or a == c:
        resultado.config(text="Isósceles")
    else:
        resultado.config(text="Escaleno")

ventana = tk.Tk()
ventana.title("Tipo de triángulo")

tk.Label(ventana, text="Lado 1:").pack()
lado1 = tk.Entry(ventana)
lado1.pack()
tk.Label(ventana, text="Lado 2:").pack()
lado2 = tk.Entry(ventana)
lado2.pack()
tk.Label(ventana, text="Lado 3:").pack()
lado3 = tk.Entry(ventana)
lado3.pack()
tk.Button(ventana, text="Clasificar", command=clasificar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Se evalúan las relaciones de igualdad entre los lados del triángulo.

---

14. Verificar si hay suficientes fondos para una compra

```
import tkinter as tk
```

```
def verificar():
    saldo = float(entrada_saldo.get())
    precio = float(entrada_precio.get())
    if saldo >= precio:
        resultado.config(text="Compra posible")
    else:
        resultado.config(text="Fondos insuficientes")

ventana = tk.Tk()
ventana.title("Verificar compra")

tk.Label(ventana, text="Saldo disponible:").pack()
entrada_saldo = tk.Entry(ventana)
entrada_saldo.pack()
tk.Label(ventana, text="Precio del artículo:").pack()
entrada_precio = tk.Entry(ventana)
entrada_precio.pack()
tk.Button(ventana, text="Verificar", command=verificar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Se evalúa si el saldo es suficiente para cubrir el costo.

---

15. Evaluar si un texto contiene una palabra clave

```
import tkinter as tk

def verificar():
    texto = entrada.get().lower()
    if "python" in texto:
        resultado.config(text="Contiene la palabra clave")
    else:
        resultado.config(text="No contiene la palabra clave")

ventana = tk.Tk()
ventana.title("Buscar palabra clave")
```

```
tk.Label(ventana, text="Texto:").pack()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Verificar", command=verificar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Se verifica si el texto ingresado contiene una palabra específica.

Programa 16: Verificar si un año es bisiesto

```
import tkinter as tk

def verificar():
    año = int(entrada.get())
    if (año % 4 == 0 and año % 100 != 0) or (año % 400 == 0):
        resultado.config(text="Año bisiesto")
    else:
        resultado.config(text="No es bisiesto")

ventana = tk.Tk()
ventana.title("Año bisiesto")

tk.Label(ventana, text="Año:").pack()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Verificar", command=verificar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Programa 17: Verificar si una palabra es un palíndromo

```
import tkinter as tk

def verificar():
```

```
    texto = entrada.get()
    if texto == texto[::-1]:
        resultado.config(text="Es un palíndromo")
    else:
        resultado.config(text="No es un palíndromo")

ventana = tk.Tk()
ventana.title("Verificar palíndromo")

tk.Label(ventana, text="Texto:").pack()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Verificar", command=verificar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Programa 18: Clasificar un número como Bajo, Medio o Alto

```
import tkinter as tk

def verificar():
    num = int(entrada.get())
    if num in [1, 2, 3]:
        resultado.config(text="Bajo")
    elif num in [4, 5, 6]:
        resultado.config(text="Medio")
    else:
        resultado.config(text="Alto")

ventana = tk.Tk()
ventana.title("Nivel del número")

tk.Label(ventana, text="Número (1-9):").pack()
entrada = tk.Entry(ventana)
entrada.pack()
```

```
tk.Button(ventana, text="Verificar", command=verificar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Programa 19: Verificar formato correcto de un nombre (inicial en mayúscula)

```
import tkinter as tk

def verificar():
    nombre = entrada.get()
    if nombre.istitle():
        resultado.config(text="Nombre con formato correcto")
    else:
        resultado.config(text="Corrige el formato del nombre")

ventana = tk.Tk()
ventana.title("Formato de nombre")

tk.Label(ventana, text="Nombre:").pack()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Verificar", command=verificar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Programa 20: Clasificación por edad

```
import tkinter as tk

def verificar():
    edad = int(entrada.get())
    if edad < 12:
        resultado.config(text="Niño")
    elif edad <= 17:
```

```
        resultado.config(text="Adolescente")
    elif edad <= 64:
        resultado.config(text="Adulto")
    else:
        resultado.config(text="Adulto mayor")

ventana = tk.Tk()
ventana.title("Categoría por edad")

tk.Label(ventana, text="Edad:").pack()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Clasificar", command=verificar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

---

3. ESTRUCTURAS REPETITIVAS. BUCLES

Ejercicio 1
Enunciado: Mostrar los números del 1 al 10 en una etiqueta.
Código:

```
import tkinter as tk

root = tk.Tk()
text = ""
for i in range(1, 11):
    text += str(i) + " "
label = tk.Label(root, text=text)
label.pack()
root.mainloop()
```

Explicación: El bucle `for` recorre del 1 al 10 y acumula los números en una cadena que se muestra con `Label`.

Ejercicio 2
Enunciado: Mostrar los números pares del 2 al 20.
Código:

```
import tkinter as tk

root = tk.Tk()
text = ""
for i in range(2, 21, 2):
    text += str(i) + " "
tk.Label(root, text=text).pack()
root.mainloop()
```

Explicación: Se usa `range(2, 21, 2)` para generar números pares.

Ejercicio 3
Enunciado: Contar hacia atrás del 10 al 1.
Código:

```
import tkinter as tk

root = tk.Tk()
text = ""
for i in range(10, 0, -1):
    text += str(i) + " "
tk.Label(root, text=text).pack()
root.mainloop()
```

Explicación: Se usa `range` con paso -1 para contar en reversa.

Ejercicio 4
Enunciado: Mostrar 10 veces "Hola".
Código:

```
import tkinter as tk

root = tk.Tk()
text = ""
for _ in range(10):
    text += "Hola\n"
tk.Label(root, text=text).pack()
root.mainloop()
```

Explicación: Se repite la palabra "Hola" 10 veces con salto de línea.

Ejercicio 5
Enunciado: Mostrar la suma de los números del 1 al 100.
Código:

```
import tkinter as tk

root = tk.Tk()
suma = 0
for i in range(1, 101):
    suma += i
tk.Label(root, text="Suma: " + str(suma)).pack()
root.mainloop()
```

Explicación: Se usa un bucle para sumar los números y se muestra el resultado.

Ejercicio 6
Enunciado: Contar cuántos múltiplos de 3 hay entre 1 y 100.
Código:

```
import tkinter as tk

root = tk.Tk()
contador = 0
for i in range(1, 101):
    if i % 3 == 0:
        contador += 1
tk.Label(root, text="Múltiplos de 3: " + str(contador)).pack()
root.mainloop()
```

Explicación: Se usa una condición dentro del bucle para contar múltiplos de 3.

Ejercicio 7
Enunciado: Mostrar una tabla de multiplicar del 5.
Código:

```
import tkinter as tk

root = tk.Tk()
text = ""
for i in range(1, 11):
    text += f"5 x {i} = {5*i}\n"
tk.Label(root, text=text).pack()
root.mainloop()
```

Explicación: El bucle genera los productos de la tabla del 5.

Ejercicio 8
Enunciado: Repetir una contraseña ingresada 3 veces.
Código:

```
import tkinter as tk

def repetir():
    text = ""
    pwd = entry.get()
    for _ in range(3):
        text += pwd + "\n"
    label.config(text=text)

root = tk.Tk()
entry = tk.Entry(root)
entry.pack()
btn = tk.Button(root, text="Repetir", command=repetir)
btn.pack()
label = tk.Label(root)
label.pack()
root.mainloop()
```

Explicación: El botón ejecuta un bucle que repite la entrada 3 veces.

Ejercicio 9
Enunciado: Mostrar los cuadrados del 1 al 10.
Código:

```
import tkinter as tk

root = tk.Tk()
text = ""
for i in range(1, 11):
    text += f"{i}^2 = {i**2}\n"
```

```
tk.Label(root, text=text).pack()
root.mainloop()
```

Explicación: El bucle eleva al cuadrado los números del 1 al 10.

Ejercicio 10
Enunciado: Mostrar una cuenta regresiva con botón.
Código:

```
import tkinter as tk

def contar():
    text = ""
    for i in range(10, 0, -1):
        text += str(i) + "\n"
    label.config(text=text)

root = tk.Tk()
btn = tk.Button(root, text="Contar", command=contar)
btn.pack()
label = tk.Label(root)
label.pack()
root.mainloop()
```

Explicación: Al pulsar el botón se inicia un bucle con cuenta atrás.

Ejercicio 11
Enunciado: Sumar los dígitos del 12345.
Código:

```
import tkinter as tk

root = tk.Tk()
numero = 12345
suma = 0
for d in str(numero):
    suma += int(d)
tk.Label(root, text="Suma de dígitos: " + str(suma)).pack()
root.mainloop()
```

Explicación: Se convierte el número en cadena y se suman sus dígitos.

Ejercicio 12
Enunciado: Mostrar el factorial de un número ingresado.
Código:

```
import tkinter as tk
```

```
def calcular():
    n = int(entry.get())
    f = 1
    for i in range(1, n+1):
        f *= i
    label.config(text="Factorial: " + str(f))

root = tk.Tk()
entry = tk.Entry(root)
entry.pack()
tk.Button(root, text="Calcular", command=calcular).pack()
label = tk.Label(root)
label.pack()
root.mainloop()
```

Explicación: El botón calcula el factorial usando un bucle.

Ejercicio 13
Enunciado: Mostrar números impares del 1 al 19.
Código:

```
import tkinter as tk

root = tk.Tk()
text = ""
for i in range(1, 20, 2):
    text += str(i) + " "
tk.Label(root, text=text).pack()
root.mainloop()
```

Explicación: Se usa paso 2 para generar impares desde 1 hasta 19.

Ejercicio 14
Enunciado: Crear una lista con nombres repetidos 5 veces.
Código:

```
import tkinter as tk

root = tk.Tk()
nombres = ["Ana", "Luis", "Sofía"]
text = ""
for nombre in nombres:
    for _ in range(5):
        text += nombre + "\n"
tk.Label(root, text=text).pack()
root.mainloop()
```

Explicación: Doble bucle: se repite cada nombre 5 veces.

Ejercicio 15
Enunciado: Generar la serie Fibonacci hasta el 10º término.
Código:

```
import tkinter as tk

root = tk.Tk()
a, b = 0, 1
text = "0 1 "
for _ in range(8):
    a, b = b, a + b
    text += str(b) + " "
tk.Label(root, text=text).pack()
root.mainloop()
```

Explicación: Se genera la serie Fibonacci con 10 términos.

Ejercicio 16
Enunciado: Mostrar 10 líneas con asteriscos crecientes.
Código:

```
import tkinter as tk

root = tk.Tk()
text = ""
for i in range(1, 11):
    text += "*" * i + "\n"
tk.Label(root, text=text).pack()
root.mainloop()
```

Explicación: Cada línea añade un asterisco más usando `* i`.

Ejercicio 17
Enunciado: Mostrar números del 100 al 50 de 5 en 5.
Código:

```
import tkinter as tk

root = tk.Tk()
text = ""
for i in range(100, 49, -5):
    text += str(i) + " "
tk.Label(root, text=text).pack()
root.mainloop()
```

Explicación: El bucle cuenta en retroceso con pasos de -5.

Ejercicio 18
Enunciado: Sumar los múltiplos de 4 del 1 al 100.
Código:

```
import tkinter as tk

root = tk.Tk()
suma = 0
for i in range(1, 101):
    if i % 4 == 0:
        suma += i
tk.Label(root, text="Suma: " + str(suma)).pack()
root.mainloop()
```

Explicación: Se comprueba si el número es múltiplo de 4 y se suma.

Ejercicio 19
Enunciado: Mostrar la cantidad de letras en cada palabra ingresada.
Código:

```
import tkinter as tk

def contar():
    palabras = entry.get().split()
    text = ""
    for p in palabras:
        text += f"{p}: {len(p)}\n"
    label.config(text=text)

root = tk.Tk()
entry = tk.Entry(root)
entry.pack()
tk.Button(root, text="Contar letras", command=contar).pack()
label = tk.Label(root)
label.pack()
root.mainloop()
```

Explicación: Se divide el texto y se muestra la longitud de cada palabra.

Ejercicio 20
Enunciado: Repetir un mensaje X veces, ingresando un número.
Código:

```
import tkinter as tk

def repetir():
    veces = int(entry.get())
    text = ""
    for _ in range(veces):
```

```
        text += "Mensaje repetido\n"
    label.config(text=text)

root = tk.Tk()
entry = tk.Entry(root)
entry.pack()
tk.Button(root, text="Repetir", command=repetir).pack()
label = tk.Label(root)
label.pack()
root.mainloop()
```

Explicación: Se repite una frase la cantidad de veces indicada por el usuario.

---

4. ESTRUCTURAS DE DATOS, LISTS

Programa 1
Descripción: Mostrar los elementos de una lista en una etiqueta.
Código:

```
import tkinter as tk

def mostrar_lista():
    lista = ["manzana", "banana", "cereza"]
    etiqueta.config(text=", ".join(lista))

ventana = tk.Tk()
boton = tk.Button(ventana, text="Mostrar frutas",
command=mostrar_lista)
etiqueta = tk.Label(ventana)
boton.pack()
etiqueta.pack()
ventana.mainloop()
```

Explicación: Se define una lista de frutas y se muestra su contenido en la etiqueta al hacer clic.

Programa 2
Descripción: Agregar elementos a una lista desde una entrada.
Código:

```
import tkinter as tk

lista = []

def agregar():
    valor = entrada.get()
    lista.append(valor)
    etiqueta.config(text=", ".join(lista))
    entrada.delete(0, tk.END)

ventana = tk.Tk()
entrada = tk.Entry(ventana)
boton = tk.Button(ventana, text="Agregar", command=agregar)
etiqueta = tk.Label(ventana)
entrada.pack()
boton.pack()
etiqueta.pack()
ventana.mainloop()
```

Explicación: El usuario escribe un elemento que se añade a la lista y se actualiza la etiqueta.

Programa 3
Descripción: Eliminar un elemento de la lista.
Código:

```
import tkinter as tk

lista = ["uno", "dos", "tres"]

def eliminar():
    if entrada.get() in lista:
        lista.remove(entrada.get())
    etiqueta.config(text=", ".join(lista))

ventana = tk.Tk()
entrada = tk.Entry(ventana)
boton = tk.Button(ventana, text="Eliminar", command=eliminar)
etiqueta = tk.Label(ventana, text=", ".join(lista))
entrada.pack()
boton.pack()
etiqueta.pack()
ventana.mainloop()
```

Explicación: Si el texto de entrada está en la lista, se elimina al presionar el botón.

Programa 4
Descripción: Buscar un elemento en la lista.
Código:

```
import tkinter as tk

lista = ["sol", "luna", "estrella"]

def buscar():
    if entrada.get() in lista:
        etiqueta.config(text="Existe")
    else:
        etiqueta.config(text="No encontrado")

ventana = tk.Tk()
entrada = tk.Entry(ventana)
boton = tk.Button(ventana, text="Buscar", command=buscar)
etiqueta = tk.Label(ventana)
entrada.pack()
boton.pack()
etiqueta.pack()
ventana.mainloop()
```

Explicación: Comprueba si el elemento ingresado existe en la lista y muestra el resultado.

Programa 5
Descripción: Contar los elementos de la lista.
Código:

```
import tkinter as tk

lista = ["a", "b", "c", "d"]

def contar():
    etiqueta.config(text=f"Número de elementos: {len(lista)}")

ventana = tk.Tk()
boton = tk.Button(ventana, text="Contar", command=contar)
etiqueta = tk.Label(ventana)
boton.pack()
etiqueta.pack()
ventana.mainloop()
```

Explicación: Se muestra la cantidad de elementos de la lista.

Programa 6
Descripción: Mostrar el elemento más largo de la lista.
Código:

```
import tkinter as tk

lista = ["gato", "elefante", "pez"]

def mayor():
    largo = max(lista, key=len)
    etiqueta.config(text=f"Elemento más largo: {largo}")

ventana = tk.Tk()
boton = tk.Button(ventana, text="Elemento largo", command=mayor)
etiqueta = tk.Label(ventana)
boton.pack()
etiqueta.pack()
ventana.mainloop()
```

Explicación: Usa `max()` con `key=len` para mostrar el string más largo.

Programa 7
Descripción: Limpiar toda la lista.
Código:

```
import tkinter as tk

lista = ["rojo", "verde", "azul"]
```

```
def limpiar():
    lista.clear()
    etiqueta.config(text="Lista vacía")

ventana = tk.Tk()
boton = tk.Button(ventana, text="Limpiar", command=limpiar)
etiqueta = tk.Label(ventana, text=", ".join(lista))
boton.pack()
etiqueta.pack()
ventana.mainloop()
```

Explicación: Vacía la lista usando `clear()`.

Programa 8
Descripción: Ordenar alfabéticamente.
Código:

```
import tkinter as tk

lista = ["banana", "pera", "manzana"]

def ordenar():
    lista.sort()
    etiqueta.config(text=", ".join(lista))

ventana = tk.Tk()
boton = tk.Button(ventana, text="Ordenar", command=ordenar)
etiqueta = tk.Label(ventana, text=", ".join(lista))
boton.pack()
etiqueta.pack()
ventana.mainloop()
```

Explicación: Usa `sort()` para ordenar la lista alfabéticamente.

Programa 9
Descripción: Mostrar lista invertida.
Código:

```
import tkinter as tk

lista = ["uno", "dos", "tres"]

def invertir():
    etiqueta.config(text=", ".join(lista[::-1]))

ventana = tk.Tk()
boton = tk.Button(ventana, text="Invertir", command=invertir)
etiqueta = tk.Label(ventana)
boton.pack()
```

```
etiqueta.pack()
ventana.mainloop()
```

Explicación: Se usa slicing para invertir la lista.

Programa 10
Descripción: Mostrar los elementos únicos.
Código:

```
import tkinter as tk

lista = ["a", "b", "a", "c", "b"]

def unicos():
    unicos = list(set(lista))
    etiqueta.config(text=", ".join(unicos))

ventana = tk.Tk()
boton = tk.Button(ventana, text="Únicos", command=unicos)
etiqueta = tk.Label(ventana)
boton.pack()
etiqueta.pack()
ventana.mainloop()
```

Explicación: `set()` elimina duplicados, y se convierte de nuevo a lista para mostrar.

Programa 11
Descripción: Mostrar la suma de una lista de números.
Código:

```
import tkinter as tk


lista = [3, 5, 8, 2]


def suma():
    total = sum(lista)
    etiqueta.config(text=f"Suma total: {total}")


ventana = tk.Tk()
boton = tk.Button(ventana, text="Sumar", command=suma)
etiqueta = tk.Label(ventana)
boton.pack()
```

```
etiqueta.pack()
ventana.mainloop()
```

Explicación: `sum(lista)` calcula la suma de todos los números.

Programa 12
Descripción: Mostrar el valor máximo de la lista.
Código:

```
import tkinter as tk

lista = [10, 50, 30, 70]

def maximo():
    etiqueta.config(text=f"Máximo: {max(lista)}")

ventana = tk.Tk()
boton = tk.Button(ventana, text="Máximo", command=maximo)
etiqueta = tk.Label(ventana)
boton.pack()
etiqueta.pack()
ventana.mainloop()
```

Explicación: `max(lista)` devuelve el número mayor de la lista.

Programa 13
Descripción: Mostrar el valor mínimo de la lista.
Código:

```
import tkinter as tk

lista = [10, 50, 30, 70]

def minimo():
    etiqueta.config(text=f"Mínimo: {min(lista)}")

ventana = tk.Tk()
```

```
boton = tk.Button(ventana, text="Mínimo", command=minimo)
etiqueta = tk.Label(ventana)
boton.pack()
etiqueta.pack()
ventana.mainloop()
```

Explicación: `min(lista)` devuelve el número menor de la lista.

Programa 14
Descripción: Insertar un elemento en una posición dada.
Código:

```
import tkinter as tk

lista = ["a", "b", "c"]

def insertar():
    valor = entrada.get()
    lista.insert(1, valor)
    etiqueta.config(text=", ".join(lista))

ventana = tk.Tk()
entrada = tk.Entry(ventana)
boton = tk.Button(ventana, text="Insertar en pos 1",
command=insertar)
etiqueta = tk.Label(ventana)
entrada.pack()
boton.pack()
etiqueta.pack()
ventana.mainloop()
```

Explicación: `insert(1, valor)` agrega un valor en la posición 1 de la lista.

Programa 15
Descripción: Contar ocurrencias de un valor en la lista.
Código:

```
import tkinter as tk

lista = ["a", "b", "a", "c", "a"]

def contar():
    valor = entrada.get()
    cantidad = lista.count(valor)
    etiqueta.config(text=f"Ocurre {cantidad} veces")

ventana = tk.Tk()
entrada = tk.Entry(ventana)
boton = tk.Button(ventana, text="Contar", command=contar)
etiqueta = tk.Label(ventana)
entrada.pack()
boton.pack()
etiqueta.pack()
ventana.mainloop()
```

Explicación: `count(valor)` devuelve cuántas veces aparece un valor en la lista.

Programa 16
Descripción: Combinar dos listas y mostrar el resultado.
Código:

```
import tkinter as tk

lista1 = ["x", "y"]
lista2 = ["a", "b"]

def combinar():
    combinado = lista1 + lista2
    etiqueta.config(text=", ".join(combinado))

ventana = tk.Tk()
boton = tk.Button(ventana, text="Combinar", command=combinar)
```

```
etiqueta = tk.Label(ventana)
boton.pack()
etiqueta.pack()
ventana.mainloop()
```

Explicación: La suma de listas (+) combina los elementos de ambas.

Programa 17
Descripción: Reemplazar un valor por otro.
Código:

```
import tkinter as tk

lista = ["perro", "gato", "pez"]

def reemplazar():
    viejo = entrada1.get()
    nuevo = entrada2.get()
    if viejo in lista:
        i = lista.index(viejo)
        lista[i] = nuevo
    etiqueta.config(text=", ".join(lista))

ventana = tk.Tk()
entrada1 = tk.Entry(ventana)
entrada2 = tk.Entry(ventana)
boton = tk.Button(ventana, text="Reemplazar", command=reemplazar)
etiqueta = tk.Label(ventana)
entrada1.pack()
entrada2.pack()
boton.pack()
etiqueta.pack()
ventana.mainloop()
```

Explicación: Busca el índice del valor antiguo y lo reemplaza con uno nuevo.

Programa 18
Descripción: Mostrar solo los valores mayores que 5.
Código:

```
import tkinter as tk

lista = [2, 7, 1, 9, 4, 10]

def filtrar():
    resultado = [str(x) for x in lista if x > 5]
    etiqueta.config(text=", ".join(resultado))

ventana = tk.Tk()
boton = tk.Button(ventana, text=">5", command=filtrar)
etiqueta = tk.Label(ventana)
boton.pack()
etiqueta.pack()
ventana.mainloop()
```

Explicación: Crea una nueva lista con los valores mayores que 5 usando comprensión.

Programa 19
Descripción: Duplicar todos los elementos numéricos de una lista.
Código:

```
import tkinter as tk

lista = [1, 2, 3, 4]

def duplicar():
    resultado = [str(x*2) for x in lista]
    etiqueta.config(text=", ".join(resultado))

ventana = tk.Tk()
boton = tk.Button(ventana, text="Duplicar", command=duplicar)
etiqueta = tk.Label(ventana)
```

```
boton.pack()
etiqueta.pack()
ventana.mainloop()
```

Explicación: Se usa una lista por comprensión para multiplicar todos los elementos por 2.

Programa 20
Descripción: Convertir una lista de números a texto.
Código:

```
import tkinter as tk

lista = [1, 2, 3, 4]

def convertir():
    texto = list(map(str, lista))
    etiqueta.config(text=", ".join(texto))

ventana = tk.Tk()
boton = tk.Button(ventana, text="Convertir", command=convertir)
etiqueta = tk.Label(ventana)
boton.pack()
etiqueta.pack()
ventana.mainloop()
```

Explicación: Se usa `map(str, lista)` para convertir todos los elementos en cadenas.

---

5. ESTRUCTURAS DE DATOS, CONJUNTOS

Ejercicio 1
Título: Crear un conjunto y mostrarlo
Explicación: Se crea un conjunto desde una lista y se muestra en una etiqueta.
Solución:

```
import tkinter as tk

def mostrar_conjunto():
    lista = [1, 2, 2, 3, 4]
    conjunto = set(lista)
    resultado.config(text=f"Conjunto: {conjunto}")

ventana = tk.Tk()
tk.Button(ventana, text="Mostrar Conjunto",
command=mostrar_conjunto).pack()
resultado = tk.Label(ventana)
resultado.pack()
ventana.mainloop()
```

Ejercicio 2
Título: Agregar elemento a conjunto
Explicación: Se agrega un número a un conjunto y se muestra actualizado.
Solución:

```
import tkinter as tk

conjunto = {1, 2, 3}

def agregar():
    conjunto.add(int(entrada.get()))
    resultado.config(text=str(conjunto))

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Agregar", command=agregar).pack()
resultado = tk.Label(ventana, text=str(conjunto))
resultado.pack()
ventana.mainloop()
```

Ejercicio 3
Título: Eliminar elemento de conjunto
Explicación: Elimina un elemento si existe, sin error.
Solución:

```
import tkinter as tk

conjunto = {1, 2, 3, 4}

def eliminar():
    conjunto.discard(int(entrada.get()))
    resultado.config(text=str(conjunto))

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Eliminar", command=eliminar).pack()
resultado = tk.Label(ventana, text=str(conjunto))
resultado.pack()
ventana.mainloop()
```

Ejercicio 4
Título: Comprobar si un elemento está en el conjunto
Explicación: Indica si un número ingresado pertenece al conjunto.
Solución:

```
import tkinter as tk

conjunto = {10, 20, 30}

def comprobar():
    num = int(entrada.get())
    existe = num in conjunto
    resultado.config(text="Está" if existe else "No está")

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Comprobar", command=comprobar).pack()
resultado = tk.Label(ventana)
resultado.pack()
ventana.mainloop()
```

Ejercicio 5
Título: Unión de dos conjuntos
Explicación: Une dos conjuntos y muestra el resultado.
Solución:

```
import tkinter as tk

def unir():
    a = {1, 2, 3}
    b = {3, 4, 5}
```

```
    union = a | b
    resultado.config(text=f"Unión: {union}")

ventana = tk.Tk()
tk.Button(ventana, text="Unir conjuntos", command=unir).pack()
resultado = tk.Label(ventana)
resultado.pack()
ventana.mainloop()
```

Ejercicio 6

Título: Intersección de dos conjuntos

Explicación: Muestra la intersección entre dos conjuntos fijos.

Solución:

```
import tkinter as tk

def intersectar():
    a = {1, 2, 3}
    b = {2, 3, 4}
    interseccion = a & b
    resultado.config(text=f"Intersección: {interseccion}")

ventana = tk.Tk()
tk.Button(ventana, text="Intersección",
command=intersectar).pack()
resultado = tk.Label(ventana)
resultado.pack()
ventana.mainloop()
```

Ejercicio 7

Título: Diferencia entre dos conjuntos

Explicación: Calcula A - B

Solución:

```
import tkinter as tk

def diferencia():
    a = {1, 2, 3, 4}
    b = {2, 3}
    resultado.config(text=f"Diferencia: {a - b}")

ventana = tk.Tk()
tk.Button(ventana, text="Diferencia A - B",
command=diferencia).pack()
resultado = tk.Label(ventana)
resultado.pack()
ventana.mainloop()
```

Ejercicio 8
Título: Diferencia simétrica
Explicación: Muestra elementos que están en un conjunto u otro, pero no en ambos.
Solución:

```
import tkinter as tk

def simetrica():
    a = {1, 2, 3}
    b = {3, 4, 5}
    resultado.config(text=f"Simétrica: {a ^ b}")

ventana = tk.Tk()
tk.Button(ventana, text="Diferencia simétrica",
command=simetrica).pack()
resultado = tk.Label(ventana)
resultado.pack()
ventana.mainloop()
```

Ejercicio 9
Título: Vaciar conjunto
Explicación: Limpia el conjunto dejando un set vacío.
Solución:

```
import tkinter as tk

conjunto = {1, 2, 3}

def vaciar():
    conjunto.clear()
    resultado.config(text=str(conjunto))

ventana = tk.Tk()
tk.Button(ventana, text="Vaciar", command=vaciar).pack()
resultado = tk.Label(ventana, text=str(conjunto))
resultado.pack()
ventana.mainloop()
```

Ejercicio 10
Título: Mostrar tamaño del conjunto
Explicación: Muestra la cantidad de elementos únicos.
Solución:

```
import tkinter as tk

conjunto = {1, 2, 3, 4, 5}
```

```
def tamano():
    resultado.config(text=f"Tamaño: {len(conjunto)}")

ventana = tk.Tk()
tk.Button(ventana, text="Tamaño", command=tamano).pack()
resultado = tk.Label(ventana)
resultado.pack()
ventana.mainloop()
```

Ejercicio 11
Título: Convertir cadena a conjunto de caracteres
Explicación: Convierte una cadena ingresada en un conjunto de letras únicas.
Solución:

```
import tkinter as tk

def convertir():
    texto = entrada.get()
    caracteres = set(texto)
    resultado.config(text=str(caracteres))

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Convertir a conjunto",
command=convertir).pack()
resultado = tk.Label(ventana)
resultado.pack()
ventana.mainloop()
```

Ejercicio 12
Título: Crear conjunto desde números separados por coma
Explicación: Crea un conjunto desde una entrada tipo 1,2,3,3
Solución:

```
import tkinter as tk

def crear():
    numeros = entrada.get().split(",")
    conjunto = set(map(int, numeros))
    resultado.config(text=str(conjunto))

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Crear conjunto", command=crear).pack()
resultado = tk.Label(ventana)
```

```
resultado.pack()
ventana.mainloop()
```

Ejercicio 13
Título: Copiar conjunto
Explicación: Realiza una copia de un conjunto y la muestra.
Solución:

```
import tkinter as tk

conjunto = {4, 5, 6}

def copiar():
    copia = conjunto.copy()
    resultado.config(text=f"Copia: {copia}")

ventana = tk.Tk()
tk.Button(ventana, text="Copiar conjunto", command=copiar).pack()
resultado = tk.Label(ventana)
resultado.pack()
ventana.mainloop()
```

Ejercicio 14
Título: Subconjunto
Explicación: Verifica si un conjunto es subconjunto de otro.
Solución:

```
import tkinter as tk

a = {1, 2}
b = {1, 2, 3, 4}

def es_subconjunto():
    resultado.config(text="Sí" if a.issubset(b) else "No")

ventana = tk.Tk()
tk.Button(ventana, text="¿A es subconjunto de B?",
command=es_subconjunto).pack()
resultado = tk.Label(ventana)
resultado.pack()
ventana.mainloop()
```

Ejercicio 15
Título: Superconjunto
Explicación: Verifica si un conjunto es superconjunto de otro.
Solución:

```
import tkinter as tk

a = {1, 2, 3, 4}
b = {2, 3}

def es_superconjunto():
    resultado.config(text="Sí" if a.issuperset(b) else "No")

ventana = tk.Tk()
tk.Button(ventana, text="¿A es superconjunto de B?",
command=es_superconjunto).pack()
resultado = tk.Label(ventana)
resultado.pack()
ventana.mainloop()
```

Ejercicio 16
Título: Ver elementos comunes entre dos entradas
Explicación: Compara dos listas y muestra elementos comunes.
Solución:

```
import tkinter as tk

def comunes():
    a = set(entry1.get().split(","))
    b = set(entry2.get().split(","))
    inter = a & b
    resultado.config(text=f"Comunes: {inter}")

ventana = tk.Tk()
entry1 = tk.Entry(ventana)
entry1.pack()
entry2 = tk.Entry(ventana)
entry2.pack()
tk.Button(ventana, text="Ver comunes", command=comunes).pack()
resultado = tk.Label(ventana)
resultado.pack()
ventana.mainloop()
```

Ejercicio 17
Título: Contar elementos únicos
Explicación: Muestra cuántos elementos únicos hay en una lista ingresada.
Solución:

```
import tkinter as tk

def contar():
    elementos = entrada.get().split(",")
    conjunto = set(elementos)
```

```
    resultado.config(text=f"Únicos: {len(conjunto)}")

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Contar únicos", command=contar).pack()
resultado = tk.Label(ventana)
resultado.pack()
ventana.mainloop()
```

Ejercicio 18
Título: Mostrar conjunto ordenado
Explicación: Muestra el conjunto convertido en lista y ordenado.
Solución:

```
import tkinter as tk

conjunto = {9, 1, 5, 3}

def ordenar():
    lista = sorted(conjunto)
    resultado.config(text=f"Ordenado: {lista}")

ventana = tk.Tk()
tk.Button(ventana, text="Ordenar conjunto",
command=ordenar).pack()
resultado = tk.Label(ventana)
resultado.pack()
ventana.mainloop()
```

Ejercicio 19
Título: Encontrar el mayor del conjunto
Explicación: Muestra el número más grande en el conjunto.
Solución:

```
import tkinter as tk

conjunto = {8, 3, 6, 1}

def mayor():
    resultado.config(text=f"Mayor: {max(conjunto)}")

ventana = tk.Tk()
tk.Button(ventana, text="Mayor", command=mayor).pack()
resultado = tk.Label(ventana)
resultado.pack()
ventana.mainloop()
```

Ejercicio 20
Título: Encontrar el menor del conjunto
Explicación: Muestra el número más pequeño en el conjunto.
Solución:

```
import tkinter as tk

conjunto = {8, 3, 6, 1}

def menor():
    resultado.config(text=f"Menor: {min(conjunto)}")

ventana = tk.Tk()
tk.Button(ventana, text="Menor", command=menor).pack()
resultado = tk.Label(ventana)
resultado.pack()
ventana.mainloop()
```

6. ESTRUCTURAS DE DATOS, TUPLAS

Ejercicio 1
Mostrar una tupla de colores en una etiqueta.

```
import tkinter as tk

root = tk.Tk()
root.title("Tupla de colores")

colores = ("Rojo", "Verde", "Azul")

etiqueta = tk.Label(root, text=str(colores))
etiqueta.pack()

root.mainloop()
```

Explicación: Se define una tupla con tres colores y se muestra su contenido en una etiqueta.

Ejercicio 2
Mostrar el primer elemento de una tupla en una etiqueta.

```
import tkinter as tk

root = tk.Tk()
root.title("Primer color")

colores = ("Rojo", "Verde", "Azul")

etiqueta = tk.Label(root, text=colores[0])
etiqueta.pack()

root.mainloop()
```

Explicación: Se accede al primer elemento de la tupla con el índice 0.

Ejercicio 3
Mostrar los elementos de una tupla en etiquetas separadas.

```
import tkinter as tk

root = tk.Tk()
root.title("Colores")

colores = ("Rojo", "Verde", "Azul")

for color in colores:
```

```
    tk.Label(root, text=color).pack()

root.mainloop()
```

Explicación: Se itera sobre la tupla y se crea una etiqueta por cada color.

Ejercicio 4
Mostrar el número de elementos de una tupla.

```
import tkinter as tk

root = tk.Tk()
root.title("Tamaño de tupla")

colores = ("Rojo", "Verde", "Azul", "Amarillo")

etiqueta = tk.Label(root, text=f"Tamaño: {len(colores)}")
etiqueta.pack()

root.mainloop()
```

Explicación: Se usa `len()` para mostrar la cantidad de elementos en la tupla.

Ejercicio 5
Buscar un valor en una tupla e indicar si está presente.

```
import tkinter as tk

def buscar():
    if entrada.get() in frutas:
        resultado.config(text="Está en la tupla")
    else:
        resultado.config(text="No está en la tupla")

root = tk.Tk()
root.title("Buscar fruta")

frutas = ("Manzana", "Pera", "Banana")

entrada = tk.Entry(root)
entrada.pack()

tk.Button(root, text="Buscar", command=buscar).pack()
resultado = tk.Label(root, text="")
resultado.pack()

root.mainloop()
```

Explicación: Compara la entrada con los elementos de la tupla.

Ejercicio 6
Mostrar la suma de los números en una tupla.

```
import tkinter as tk

root = tk.Tk()
root.title("Suma de tupla")

numeros = (5, 3, 8, 1)

etiqueta = tk.Label(root, text=f"Suma: {sum(numeros)}")
etiqueta.pack()

root.mainloop()
```

Explicación: Se usa `sum()` para sumar los valores de la tupla.

Ejercicio 7
Concatenar dos tuplas y mostrar el resultado.

```
import tkinter as tk

root = tk.Tk()
root.title("Tupla combinada")

a = (1, 2, 3)
b = (4, 5, 6)
c = a + b

etiqueta = tk.Label(root, text=str(c))
etiqueta.pack()

root.mainloop()
```

Explicación: Se usa `+` para concatenar las tuplas.

Ejercicio 8
Contar cuántas veces aparece un elemento en una tupla.

```
import tkinter as tk

root = tk.Tk()
root.title("Contador")
```

```
letras = ("a", "b", "a", "c", "a")

etiqueta = tk.Label(root, text=f"'a' aparece {letras.count('a')}
veces")
etiqueta.pack()

root.mainloop()
```

Explicación: `count()` devuelve cuántas veces aparece un valor.

Ejercicio 9
Encontrar la posición de un valor en una tupla.

```
import tkinter as tk

root = tk.Tk()
root.title("Índice de valor")

valores = (10, 20, 30, 40)

etiqueta = tk.Label(root, text=f"30 está en el índice
{valores.index(30)}")
etiqueta.pack()

root.mainloop()
```

Explicación: `index()` devuelve la posición de un valor en la tupla.

Ejercicio 10
Convertir una tupla a cadena y mostrarla.

```
import tkinter as tk

root = tk.Tk()
root.title("Tupla como cadena")

datos = ("Nombre", "Edad", "Ciudad")

cadena = ", ".join(datos)

tk.Label(root, text=cadena).pack()

root.mainloop()
```

Explicación: Se convierte la tupla a una cadena con `join()`.

Ejercicio 11
Mostrar el último elemento de una tupla.

```
import tkinter as tk

root = tk.Tk()
root.title("Último elemento")

nombres = ("Ana", "Luis", "Marta")

etiqueta = tk.Label(root, text=nombres[-1])
etiqueta.pack()

root.mainloop()
```

Explicación: Se usa índice negativo para acceder al último elemento.

Ejercicio 12
Verificar si la tupla está vacía.

```
import tkinter as tk

root = tk.Tk()
root.title("¿Tupla vacía?")

datos = ()

mensaje = "Vacía" if not datos else "No vacía"

tk.Label(root, text=mensaje).pack()

root.mainloop()
```

Explicación: Se evalúa la tupla directamente en una condición.

Ejercicio 13
Invertir el orden de una tupla y mostrarla.

```
import tkinter as tk

root = tk.Tk()
root.title("Tupla invertida")

numeros = (1, 2, 3, 4, 5)

invertida = numeros[::-1]

tk.Label(root, text=str(invertida)).pack()
```

```
root.mainloop()
```

Explicación: Se usa slicing para invertir la tupla.

Ejercicio 14
Mostrar los elementos de una tupla con separación de líneas.

```
import tkinter as tk

root = tk.Tk()
root.title("Lista vertical")

lenguajes = ("Python", "Java", "C++")

texto = "\n".join(lenguajes)

tk.Label(root, text=texto).pack()

root.mainloop()
```

Explicación: Se unen los elementos con `\n` para separarlos en líneas.

Ejercicio 15
Convertir lista a tupla y mostrar.

```
import tkinter as tk

root = tk.Tk()
root.title("Lista a tupla")

lista = ["A", "B", "C"]
tupla = tuple(lista)

tk.Label(root, text=str(tupla)).pack()

root.mainloop()
```

Explicación: Se convierte una lista en tupla con `tuple()`.

Ejercicio 16
Mostrar una tupla de tuplas en una etiqueta.

```
import tkinter as tk

root = tk.Tk()
```

```
root.title("Tupla de tuplas")

datos = (("Nombre", "Ana"), ("Edad", 25))

tk.Label(root, text=str(datos)).pack()

root.mainloop()
```

Explicación: Se muestra directamente una tupla que contiene tuplas.

Ejercicio 17
Mostrar los valores de una tupla de pares clave-valor.

```
import tkinter as tk

root = tk.Tk()
root.title("Claves y valores")

tupla = (("Nombre", "Carlos"), ("Edad", 30))

for clave, valor in tupla:
    tk.Label(root, text=f"{clave}: {valor}").pack()

root.mainloop()
```

Explicación: Se desempaquetan las sub-tuplas para mostrar clave y valor.

Ejercicio 18
Un botón que muestre un valor de la tupla al azar.

```
import tkinter as tk
import random

def mostrar():
    resultado.config(text=random.choice(animales))

root = tk.Tk()
root.title("Animal al azar")

animales = ("Perro", "Gato", "Loro", "Pez")

tk.Button(root, text="Elegir", command=mostrar).pack()
resultado = tk.Label(root, text="")
resultado.pack()

root.mainloop()
```

Explicación: Se usa `random.choice()` para elegir un valor aleatorio.

Ejercicio 19
Contar cuántos elementos son mayores a 10 en una tupla.

```
import tkinter as tk

root = tk.Tk()
root.title("Mayores que 10")

numeros = (4, 12, 7, 15, 22)

mayores = len([x for x in numeros if x > 10])

tk.Label(root, text=f"{mayores} números son mayores a 10").pack()

root.mainloop()
```

Explicación: Se usa comprensión de listas para contar valores.

Ejercicio 20
Mostrar si una tupla contiene solo números.

```
import tkinter as tk

root = tk.Tk()
root.title("¿Solo números?")

valores = (3, 6, "Hola", 9)

solo_numeros = all(isinstance(x, (int, float)) for x in valores)

mensaje = "Todos son números" if solo_numeros else "Hay otros
tipos"

tk.Label(root, text=mensaje).pack()

root.mainloop()
```

Explicación: Se usa `all()` y `isinstance()` para verificar tipos.

---

7. ESTRUCTURAS DE DATOS, DICCIONARIOS

Ejercicio 1
Mostrar el valor de una clave ingresada por el usuario.

Explicación: Se utiliza un Entry para ingresar la clave y se muestra su valor si existe.

Solución:

```
import tkinter as tk

datos = {'nombre': 'Ana', 'edad': 30, 'ciudad': 'Zaragoza'}

def buscar():
    clave = entrada.get()
    valor = datos.get(clave, 'Clave no encontrada')
    resultado.config(text=valor)

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Buscar", command=buscar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()
ventana.mainloop()
```

Ejercicio 2
Agregar un par clave-valor al diccionario.

Explicación: El usuario introduce una clave y un valor, que se agregan al diccionario.

Solución:

```
import tkinter as tk

datos = {}

def agregar():
    clave = clave_entry.get()
    valor = valor_entry.get()
    datos[clave] = valor
    resultado.config(text=str(datos))

ventana = tk.Tk()
clave_entry = tk.Entry(ventana)
clave_entry.pack()
valor_entry = tk.Entry(ventana)
valor_entry.pack()
tk.Button(ventana, text="Agregar", command=agregar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()
ventana.mainloop()
```

Ejercicio 3
Eliminar una clave del diccionario.

Explicación: Se elimina la clave que introduce el usuario.

Solución:

```
import tkinter as tk

datos = {'a': 1, 'b': 2, 'c': 3}

def eliminar():
    clave = entrada.get()
    datos.pop(clave, None)
    resultado.config(text=str(datos))

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Eliminar", command=eliminar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()
ventana.mainloop()
```

Ejercicio 4
Verificar si una clave está en el diccionario.

Explicación: Se usa `in` para comprobar si la clave existe.

Solución:

```
import tkinter as tk

datos = {'a': 1, 'b': 2}

def verificar():
    clave = entrada.get()
    existe = clave in datos
    resultado.config(text=str(existe))

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Verificar", command=verificar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()
ventana.mainloop()
```

Ejercicio 5
Mostrar todas las claves del diccionario.

Explicación: Se llama a `keys()` y se muestra en pantalla.

Solución:

```
import tkinter as tk

datos = {'uno': 1, 'dos': 2, 'tres': 3}

def mostrar_claves():
    claves = list(datos.keys())
    resultado.config(text=str(claves))

ventana = tk.Tk()
tk.Button(ventana, text="Mostrar claves",
command=mostrar_claves).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()
ventana.mainloop()
```

Ejercicio 6
Mostrar todos los valores del diccionario.

Explicación: Usa `values()` para obtener los valores.

Solución:

```
import tkinter as tk

datos = {'x': 10, 'y': 20, 'z': 30}

def mostrar_valores():
    valores = list(datos.values())
    resultado.config(text=str(valores))

ventana = tk.Tk()
tk.Button(ventana, text="Mostrar valores",
command=mostrar_valores).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()
ventana.mainloop()
```

Ejercicio 7
Mostrar todos los elementos (items) del diccionario.

Explicación: Se listan los pares clave-valor.

Solución:

```
import tkinter as tk

datos = {'rojo': '#f00', 'verde': '#0f0'}

def mostrar_items():
    items = list(datos.items())
    resultado.config(text=str(items))

ventana = tk.Tk()
tk.Button(ventana, text="Mostrar items",
command=mostrar_items).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()
ventana.mainloop()
```

Ejercicio 8
Contar cuántos elementos tiene el diccionario.

Explicación: Usa `len()` para obtener el tamaño.

Solución:

```
import tkinter as tk

datos = {'a': 1, 'b': 2}

def contar():
    total = len(datos)
    resultado.config(text=str(total))

ventana = tk.Tk()
tk.Button(ventana, text="Contar", command=contar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()
ventana.mainloop()
```

Ejercicio 9
Limpiar el diccionario.

Explicación: Se llama a `clear()`.

Solución:

```
import tkinter as tk

datos = {'uno': 1, 'dos': 2}

def limpiar():
    datos.clear()
    resultado.config(text=str(datos))
```

```
ventana = tk.Tk()
tk.Button(ventana, text="Limpiar", command=limpiar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()
ventana.mainloop()
```

Ejercicio 10
Actualizar un valor existente en el diccionario.

Explicación: Se accede a la clave y se asigna un nuevo valor.

Solución:

```
import tkinter as tk

datos = {'clave': 'valor'}

def actualizar():
    clave = entrada_clave.get()
    valor = entrada_valor.get()
    if clave in datos:
        datos[clave] = valor
    resultado.config(text=str(datos))

ventana = tk.Tk()
entrada_clave = tk.Entry(ventana)
entrada_clave.pack()
entrada_valor = tk.Entry(ventana)
entrada_valor.pack()
tk.Button(ventana, text="Actualizar", command=actualizar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()
ventana.mainloop()
```

Ejercicio 11
Crear un diccionario con nombres y edades desde la interfaz.

Explicación: Se ingresan pares nombre-edad y se agregan al diccionario.

Solución:

```
import tkinter as tk

datos = {}

def agregar():
    nombre = entrada_nombre.get()
    edad = entrada_edad.get()
    if nombre and edad:
```

```
        datos[nombre] = int(edad)
    resultado.config(text=str(datos))

ventana = tk.Tk()
entrada_nombre = tk.Entry(ventana)
entrada_nombre.pack()
entrada_edad = tk.Entry(ventana)
entrada_edad.pack()
tk.Button(ventana, text="Agregar persona", command=agregar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()
ventana.mainloop()
```

Ejercicio 12
Mostrar la edad de una persona a partir de su nombre.

Explicación: Se busca el nombre y se muestra la edad almacenada.

Solución:

```
import tkinter as tk

datos = {'Ana': 25, 'Luis': 30}

def mostrar_edad():
    nombre = entrada.get()
    edad = datos.get(nombre, 'No encontrado')
    resultado.config(text=edad)

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Mostrar edad",
command=mostrar_edad).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()
ventana.mainloop()
```

Ejercicio 13
Sumar todos los valores numéricos del diccionario.

Explicación: Se suman todos los valores usando `sum()`.

Solución:

```
import tkinter as tk

datos = {'a': 10, 'b': 20, 'c': 30}

def sumar_valores():
```

```
    total = sum(datos.values())
    resultado.config(text=str(total))

ventana = tk.Tk()
tk.Button(ventana, text="Sumar valores",
command=sumar_valores).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()
ventana.mainloop()
```

Ejercicio 14
Obtener el valor máximo del diccionario.

Explicación: Usa `max()` sobre los valores del diccionario.

Solución:

```
import tkinter as tk

datos = {'a': 5, 'b': 9, 'c': 3}

def valor_maximo():
    if datos:
        maximo = max(datos.values())
    else:
        maximo = 'Diccionario vacío'
    resultado.config(text=str(maximo))

ventana = tk.Tk()
tk.Button(ventana, text="Valor máximo",
command=valor_maximo).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()
ventana.mainloop()
```

Ejercicio 15
Copiar un diccionario a otro.

Explicación: Se usa `copy()` para clonar el diccionario.

Solución:

```
import tkinter as tk

original = {'x': 1, 'y': 2}
copia = {}

def copiar():
    global copia
    copia = original.copy()
```

```
    resultado.config(text=str(copia))

ventana = tk.Tk()
tk.Button(ventana, text="Copiar diccionario",
command=copiar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()
ventana.mainloop()
```

Ejercicio 16
Crear un diccionario de conteo de palabras.

Explicación: Se cuenta cuántas veces aparece cada palabra ingresada.

Solución:

```
import tkinter as tk

conteo = {}

def contar():
    palabra = entrada.get()
    conteo[palabra] = conteo.get(palabra, 0) + 1
    resultado.config(text=str(conteo))

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Contar palabra", command=contar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()
ventana.mainloop()
```

Ejercicio 17
Filtrar claves que empiecen por una letra.

Explicación: Se usa comprensión de diccionario para filtrar claves.

Solución:

```
import tkinter as tk

datos = {'Ana': 25, 'Alberto': 30, 'Carlos': 40}

def filtrar():
    letra = entrada.get()
    filtrado = {k: v for k, v in datos.items() if
k.startswith(letra)}
    resultado.config(text=str(filtrado))
```

```
ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Filtrar por letra",
command=filtrar).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()
ventana.mainloop()
```

Ejercicio 18
Convertir el diccionario en texto plano.

Explicación: Se genera una cadena con los elementos formateados.

Solución:

```
import tkinter as tk

datos = {'a': 1, 'b': 2, 'c': 3}

def convertir():
    texto = '\n'.join(f'{k}: {v}' for k, v in datos.items())
    resultado.config(text=texto)

ventana = tk.Tk()
tk.Button(ventana, text="Convertir a texto",
command=convertir).pack()
resultado = tk.Label(ventana, text="", justify="left")
resultado.pack()
ventana.mainloop()
```

Ejercicio 19
Revertir claves y valores del diccionario.

Explicación: Se invierte el diccionario, asumiendo valores únicos.

Solución:

```
import tkinter as tk

datos = {'x': '1', 'y': '2'}

def invertir():
    invertido = {v: k for k, v in datos.items()}
    resultado.config(text=str(invertido))

ventana = tk.Tk()
tk.Button(ventana, text="Invertir", command=invertir).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()
```

```
ventana.mainloop()
```

Ejercicio 20
Mostrar claves cuyos valores sean mayores que un número dado.

Explicación: Se filtran claves según el valor ingresado.

Solución:

```
import tkinter as tk

datos = {'a': 10, 'b': 20, 'c': 5}

def mayores_que():
    n = int(entrada.get())
    seleccion = [k for k, v in datos.items() if v > n]
    resultado.config(text=str(seleccion))

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Mostrar mayores que...",
command=mayores_que).pack()
resultado = tk.Label(ventana, text="")
resultado.pack()
ventana.mainloop()
```

---

8. FUNCIONES

1. Mostrar mensaje con función

```
import tkinter as tk

def mostrar_mensaje():
    etiqueta.config(text="Hola, mundo!")

ventana = tk.Tk()
ventana.title("Mensaje")

boton = tk.Button(ventana, text="Saludar",
command=mostrar_mensaje)
boton.pack()

etiqueta = tk.Label(ventana, text="")
etiqueta.pack()

ventana.mainloop()
```

Explicación: Se crea una función `mostrar_mensaje` que actualiza el texto de una etiqueta.

---

2. Sumar dos números

```
import tkinter as tk

def sumar():
    resultado = int(entrada1.get()) + int(entrada2.get())
    etiqueta_resultado.config(text="Resultado: " +
str(resultado))

ventana = tk.Tk()

entrada1 = tk.Entry(ventana)
entrada1.pack()

entrada2 = tk.Entry(ventana)
entrada2.pack()

boton = tk.Button(ventana, text="Sumar", command=sumar)
boton.pack()

etiqueta_resultado = tk.Label(ventana, text="")
etiqueta_resultado.pack()
```

```
ventana.mainloop()
```

Explicación: La función `sumar` obtiene los valores de las cajas de texto y muestra la suma.

---

3. Contar clics

```
import tkinter as tk

contador = 0

def contar():
    global contador
    contador += 1
    etiqueta.config(text="Clics: " + str(contador))

ventana = tk.Tk()
boton = tk.Button(ventana, text="Clic", command=contar)
boton.pack()

etiqueta = tk.Label(ventana, text="Clics: 0")
etiqueta.pack()

ventana.mainloop()
```

Explicación: La función `contar` incrementa un contador global cada vez que se presiona el botón.

---

4. Cambiar color de fondo

```
import tkinter as tk

def cambiar_color():
    ventana.config(bg="lightblue")

ventana = tk.Tk()
boton = tk.Button(ventana, text="Color",
command=cambiar_color)
boton.pack()

ventana.mainloop()
```

Explicación: La función `cambiar_color` cambia el color del fondo de la ventana.

---

5. Concatenar texto

```
import tkinter as tk

def concatenar():
    nombre = entrada.get()
    etiqueta.config(text="Hola " + nombre)

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()

boton = tk.Button(ventana, text="Decir hola",
command=concatenar)
boton.pack()

etiqueta = tk.Label(ventana, text="")
etiqueta.pack()

ventana.mainloop()
```

Explicación: La función obtiene el texto ingresado y lo concatena con un saludo.

---

6. Limpiar texto

```
import tkinter as tk

def limpiar():
    entrada.delete(0, tk.END)

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()

boton = tk.Button(ventana, text="Limpiar", command=limpiar)
boton.pack()

ventana.mainloop()
```

Explicación: La función `limpiar` borra el texto de la caja de entrada.

---

7. Verificar si es número par

```
import tkinter as tk

def verificar_par():
    numero = int(entrada.get())
    if numero % 2 == 0:
```

```
        resultado.config(text="Es par")
    else:
        resultado.config(text="Es impar")

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()

boton = tk.Button(ventana, text="Verificar",
command=verificar_par)
boton.pack()

resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Usa una función para determinar si un número ingresado es par.

---

8. Multiplicación por 10

```
import tkinter as tk

def multiplicar():
    numero = int(entrada.get())
    resultado.config(text="x10 = " + str(numero * 10))

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()

boton = tk.Button(ventana, text="Multiplicar",
command=multiplicar)
boton.pack()

resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Multiplica el número ingresado por 10.

---

9. Convertir a mayúsculas

```
import tkinter as tk
```

```
def a_mayusculas():
    texto = entrada.get()
    resultado.config(text=texto.upper())

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()

boton = tk.Button(ventana, text="Mayúsculas",
command=a_mayusculas)
boton.pack()

resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Convierte el texto a mayúsculas al hacer clic.

---

10. Calcular cuadrado

```
import tkinter as tk

def cuadrado():
    numero = int(entrada.get())
    resultado.config(text="Cuadrado: " + str(numero ** 2))

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()

boton = tk.Button(ventana, text="Cuadrado", command=cuadrado)
boton.pack()

resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Calcula el cuadrado del número ingresado.

---

11. Mostrar longitud de texto

```
import tkinter as tk

def longitud():
```

```
    texto = entrada.get()
    resultado.config(text="Longitud: " + str(len(texto)))

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()

boton = tk.Button(ventana, text="Contar", command=longitud)
boton.pack()

resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Cuenta los caracteres del texto ingresado.

---

12. Convertir a binario

```
import tkinter as tk

def a_binario():
    numero = int(entrada.get())
    resultado.config(text=bin(numero))

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()

boton = tk.Button(ventana, text="Binario", command=a_binario)
boton.pack()

resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Muestra la representación binaria del número.

---

13. Mostrar fecha actual

```
import tkinter as tk
import datetime

def mostrar_fecha():
    fecha = datetime.date.today()
```

```
    etiqueta.config(text=str(fecha))

ventana = tk.Tk()
boton = tk.Button(ventana, text="Mostrar fecha",
command=mostrar_fecha)
boton.pack()

etiqueta = tk.Label(ventana, text="")
etiqueta.pack()

ventana.mainloop()
```

Explicación: Usa la función `today` para mostrar la fecha actual.

---

14. Mostrar número aleatorio

```
import tkinter as tk
import random

def aleatorio():
    numero = random.randint(1, 100)
    resultado.config(text="Número: " + str(numero))

ventana = tk.Tk()
boton = tk.Button(ventana, text="Aleatorio",
command=aleatorio)
boton.pack()

resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Genera un número aleatorio al presionar el botón.

---

15. Determinar vocal

```
import tkinter as tk

def es_vocal():
    letra = entrada.get().lower()
    if letra in 'aeiou':
        resultado.config(text="Es vocal")
    else:
        resultado.config(text="No es vocal")
```

```
ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()

boton = tk.Button(ventana, text="Verificar",
command=es_vocal)
boton.pack()

resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Determina si el carácter ingresado es una vocal.

---

16. Calcular factorial

```
import tkinter as tk
import math

def factorial():
    numero = int(entrada.get())
    resultado.config(text="Factorial: " +
str(math.factorial(numero)))

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()

boton = tk.Button(ventana, text="Factorial",
command=factorial)
boton.pack()

resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Calcula el factorial usando el módulo `math`.

---

17. Convertir Celsius a Fahrenheit

```
import tkinter as tk

def convertir():
    c = float(entrada.get())
```

```
    f = c * 9/5 + 32
    resultado.config(text="Fahrenheit: " + str(f))

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()

boton = tk.Button(ventana, text="Convertir",
command=convertir)
boton.pack()

resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Realiza una conversión de temperatura.

---

18. Mostrar mensaje personalizado

```
import tkinter as tk

def mostrar():
    nombre = entrada.get()
    resultado.config(text=f"Bienvenido, {nombre}")

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()

boton = tk.Button(ventana, text="Entrar", command=mostrar)
boton.pack()

resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Saluda al usuario con el nombre introducido.

---

19. Calcular área de círculo

```
import tkinter as tk
import math

def calcular_area():
```

```
    radio = float(entrada.get())
    area = math.pi * radio ** 2
    resultado.config(text="Área: " + str(round(area, 2)))

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()

boton = tk.Button(ventana, text="Área círculo",
command=calcular_area)
boton.pack()

resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Calcula el área del círculo con el radio ingresado.

---

20. Invertir texto

```
import tkinter as tk

def invertir():
    texto = entrada.get()
    resultado.config(text=texto[::-1])

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()

boton = tk.Button(ventana, text="Invertir", command=invertir)
boton.pack()

resultado = tk.Label(ventana, text="")
resultado.pack()

ventana.mainloop()
```

Explicación: Invierte el texto ingresado en la caja.

---

9. PROGRAMACION ORIENTADA A OBJETOS

Ejercicio 1
Título: Ventana básica con botón
Explicación: Se crea una clase `VentanaBasica` con un botón que cierra la ventana al pulsarse.
Solución:

```
import tkinter as tk

class VentanaBasica:
    def __init__(self):
        self.ventana = tk.Tk()
        self.ventana.title("Ejercicio 1")
        self.boton = tk.Button(self.ventana, text="Cerrar",
command=self.ventana.quit)
        self.boton.pack()
        self.ventana.mainloop()

VentanaBasica()
```

Ejercicio 2
Título: Entrada de texto y etiqueta
Explicación: Al escribir en la entrada y pulsar el botón, se muestra el texto en una etiqueta.
Solución:

```
import tkinter as tk

class MostrarTexto:
    def __init__(self):
        self.ventana = tk.Tk()
        self.entrada = tk.Entry(self.ventana)
        self.entrada.pack()
        self.boton = tk.Button(self.ventana, text="Mostrar",
command=self.mostrar)
        self.boton.pack()
        self.etiqueta = tk.Label(self.ventana, text="")
        self.etiqueta.pack()
        self.ventana.mainloop()

    def mostrar(self):
        texto = self.entrada.get()
        self.etiqueta.config(text=texto)

MostrarTexto()
```

Ejercicio 3
Título: Contador simple

Explicación: Un botón aumenta el número mostrado en una etiqueta.
Solución:

```
import tkinter as tk

class Contador:
    def __init__(self):
        self.ventana = tk.Tk()
        self.contador = 0
        self.etiqueta = tk.Label(self.ventana, text="0")
        self.etiqueta.pack()
        self.boton = tk.Button(self.ventana, text="Incrementar",
command=self.incrementar)
        self.boton.pack()
        self.ventana.mainloop()

    def incrementar(self):
        self.contador += 1
        self.etiqueta.config(text=str(self.contador))

Contador()
```

Ejercicio 4
Título: Cambiar color de fondo
Explicación: Cambia el color del fondo al pulsar el botón.
Solución:

```
import tkinter as tk

class CambiarColor:
    def __init__(self):
        self.ventana = tk.Tk()
        self.boton = tk.Button(self.ventana, text="Cambiar color",
command=self.cambiar)
        self.boton.pack()
        self.ventana.mainloop()

    def cambiar(self):
        self.ventana.configure(bg="lightblue")

CambiarColor()
```

Ejercicio 5
Título: Selector de opciones
Explicación: Usa `Radiobutton` para elegir una opción y mostrarla.
Solución:

```
import tkinter as tk
```

```
class SelectorOpciones:
    def __init__(self):
        self.ventana = tk.Tk()
        self.opcion = tk.StringVar()
        self.opcion.set("A")
        tk.Radiobutton(self.ventana, text="Opción A",
variable=self.opcion, value="A").pack()
        tk.Radiobutton(self.ventana, text="Opción B",
variable=self.opcion, value="B").pack()
        tk.Button(self.ventana, text="Mostrar",
command=self.mostrar).pack()
        self.etiqueta = tk.Label(self.ventana, text="")
        self.etiqueta.pack()
        self.ventana.mainloop()

    def mostrar(self):
        self.etiqueta.config(text="Seleccionaste: " +
self.opcion.get())

SelectorOpciones()
```

Ejercicio 6
Título: Casillas de verificación
Explicación: Muestra qué casillas están seleccionadas.
Solución:

```
import tkinter as tk


class Verificacion:
    def __init__(self):
        self.ventana = tk.Tk()
        self.var1 = tk.IntVar()
        self.var2 = tk.IntVar()
        tk.Checkbutton(self.ventana, text="Opción 1", variable=self.var1).pack()
        tk.Checkbutton(self.ventana, text="Opción 2", variable=self.var2).pack()
        tk.Button(self.ventana, text="Verificar", command=self.mostrar).pack()
        self.etiqueta = tk.Label(self.ventana, text="")
        self.etiqueta.pack()
        self.ventana.mainloop()


    def mostrar(self):
        resultado = "Seleccionaste: "
        if self.var1.get():
            resultado += "Opción 1 "
```

```
        if self.var2.get():
            resultado += "Opción 2"
        self.etiqueta.config(text=resultado)

Verificacion()
```

Ejercicio 7
Título: Cambiar texto de botón
Explicación: Al hacer clic, cambia el texto del botón.
Solución:

```
import tkinter as tk

class BotonDinamico:
    def __init__(self):
        self.ventana = tk.Tk()
        self.boton = tk.Button(self.ventana, text="Haz clic",
command=self.cambiar_texto)
        self.boton.pack()
        self.ventana.mainloop()

    def cambiar_texto(self):
        self.boton.config(text="¡Clic hecho!")

BotonDinamico()
```

Ejercicio 8
Título: Saludo personalizado
Explicación: Pide el nombre del usuario y lo saluda.
Solución:

```
import tkinter as tk

class Saludo:
    def __init__(self):
        self.ventana = tk.Tk()
        self.entrada = tk.Entry(self.ventana)
        self.entrada.pack()
        tk.Button(self.ventana, text="Saludar", command=self.saludar).pack()
        self.etiqueta = tk.Label(self.ventana, text="")
```

```
        self.etiqueta.pack()
        self.ventana.mainloop()

    def saludar(self):
        nombre = self.entrada.get()
        self.etiqueta.config(text="Hola, " + nombre + "!")

Saludo()
```

Ejercicio 9
Título: Suma de dos números
Explicación: Suma dos números introducidos y muestra el resultado.
Solución:

```
import tkinter as tk

class SumaNumeros:
    def __init__(self):
        self.ventana = tk.Tk()
        self.e1 = tk.Entry(self.ventana)
        self.e2 = tk.Entry(self.ventana)
        self.e1.pack()
        self.e2.pack()
        tk.Button(self.ventana, text="Sumar", command=self.sumar).pack()
        self.resultado = tk.Label(self.ventana, text="")
        self.resultado.pack()
        self.ventana.mainloop()

    def sumar(self):
        try:
            n1 = float(self.e1.get())
            n2 = float(self.e2.get())
            self.resultado.config(text="Resultado: " + str(n1 + n2))
        except:
            self.resultado.config(text="Entrada inválida")

SumaNumeros()
```

Ejercicio 10
Título: Cambiar tamaño de fuente
Explicación: Un botón aumenta el tamaño de letra de una etiqueta.
Solución:

```
import tkinter as tk


class FuenteDinamica:
    def __init__(self):
        self.ventana = tk.Tk()
        self.tamano = 10
        self.etiqueta = tk.Label(self.ventana, text="Texto dinámico",
font=("Arial", self.tamano))
        self.etiqueta.pack()
        tk.Button(self.ventana, text="Aumentar fuente",
command=self.aumentar).pack()
        self.ventana.mainloop()


    def aumentar(self):
        self.tamano += 2
        self.etiqueta.config(font=("Arial", self.tamano))


FuenteDinamica()
```

Ejercicio 11
Título: Calculadora simple
Explicación: Realiza suma, resta, multiplicación y división.
Solución:

```
import tkinter as tk


class Calculadora:
    def __init__(self):
        self.ventana = tk.Tk()
        self.e1 = tk.Entry(self.ventana)
        self.e2 = tk.Entry(self.ventana)
        self.e1.pack()
        self.e2.pack()
        tk.Button(self.ventana, text="Sumar", command=self.sumar).pack()
        tk.Button(self.ventana, text="Restar", command=self.restar).pack()
```

```
        tk.Button(self.ventana, text="Multiplicar",
command=self.multiplicar).pack()
        tk.Button(self.ventana, text="Dividir", command=self.dividir).pack()
        self.resultado = tk.Label(self.ventana, text="")
        self.resultado.pack()
        self.ventana.mainloop()

    def sumar(self):
        self.operacion(lambda a, b: a + b)

    def restar(self):
        self.operacion(lambda a, b: a - b)

    def multiplicar(self):
        self.operacion(lambda a, b: a * b)

    def dividir(self):
        self.operacion(lambda a, b: a / b if b != 0 else "División por cero")

    def operacion(self, funcion):
        try:
            a = float(self.e1.get())
            b = float(self.e2.get())
            r = funcion(a, b)
            self.resultado.config(text="Resultado: " + str(r))
        except:
            self.resultado.config(text="Entrada inválida")

Calculadora()
```

Ejercicio 12
Título: Temporizador simple
Explicación: Muestra un contador que se incrementa cada segundo usando `after`.
Solución:

```
import tkinter as tk

class Temporizador:
```

```
    def __init__(self):
        self.ventana = tk.Tk()
        self.contador = 0
        self.etiqueta = tk.Label(self.ventana, text="0")
        self.etiqueta.pack()
        self.actualizar()
        self.ventana.mainloop()

    def actualizar(self):
        self.contador += 1
        self.etiqueta.config(text=str(self.contador))
        self.ventana.after(1000, self.actualizar)

Temporizador()
```

Ejercicio 13
Título: Lista desplegable (Combobox)
Explicación: Permite seleccionar un elemento y mostrarlo.
Solución:

```
import tkinter as tk
from tkinter import ttk

class ListaDesplegable:
    def __init__(self):
        self.ventana = tk.Tk()
        self.combo = ttk.Combobox(self.ventana, values=["Rojo",
"Verde", "Azul"])
        self.combo.current(0)
        self.combo.pack()
        tk.Button(self.ventana, text="Mostrar",
command=self.mostrar).pack()
        self.etiqueta = tk.Label(self.ventana, text="")
        self.etiqueta.pack()
        self.ventana.mainloop()
```

```
    def mostrar(self):
        self.etiqueta.config(text="Seleccionaste: " +
self.combo.get())

ListaDesplegable()
```

Ejercicio 14
Título: Entrada con validación numérica
Explicación: Solo acepta números en la entrada.
Solución:

```
import tkinter as tk

class EntradaNumerica:
    def __init__(self):
        self.ventana = tk.Tk()
        vcmd = (self.ventana.register(self.validar), '%P')
        self.entrada = tk.Entry(self.ventana, validate='key',
validatecommand=vcmd)
        self.entrada.pack()
        self.ventana.mainloop()

    def validar(self, texto):
        return texto.isdigit() or texto == ""

EntradaNumerica()
```

Ejercicio 15
Título: Mostrar imagen
Explicación: Carga y muestra una imagen PNG en la ventana.
Solución:

```
import tkinter as tk
from PIL import Image, ImageTk
```

```
class MostrarImagen:
    def __init__(self):
        self.ventana = tk.Tk()
        imagen = Image.open("ruta_imagen.png")  # Cambia por ruta
válida
        self.foto = ImageTk.PhotoImage(imagen)
        self.label = tk.Label(self.ventana, image=self.foto)
        self.label.pack()
        self.ventana.mainloop()

MostrarImagen()
```

Nota: Necesitas instalar Pillow (`pip install pillow`) y tener una imagen válida.

Ejercicio 16
Título: Ventana con pestañas
Explicación: Crea pestañas con contenido diferente usando `Notebook`.
Solución:

```
import tkinter as tk
from tkinter import ttk

class VentanaPestanas:
    def __init__(self):
        self.ventana = tk.Tk()
        self.notebook = ttk.Notebook(self.ventana)
        self.p1 = ttk.Frame(self.notebook)
        self.p2 = ttk.Frame(self.notebook)
        self.notebook.add(self.p1, text="Pestaña 1")
        self.notebook.add(self.p2, text="Pestaña 2")
        self.notebook.pack(expand=True, fill='both')

        tk.Label(self.p1, text="Contenido de pestaña 1").pack()
        tk.Label(self.p2, text="Contenido de pestaña 2").pack()
```

```
        self.ventana.mainloop()

VentanaPestanas()
```

Ejercicio 17
Título: Cuadro de diálogo simple
Explicación: Muestra un mensaje al pulsar un botón.
Solución:

```
import tkinter as tk
from tkinter import messagebox

class Dialogo:
    def __init__(self):
        self.ventana = tk.Tk()
        tk.Button(self.ventana, text="Mostrar mensaje",
command=self.mensaje).pack()
        self.ventana.mainloop()

    def mensaje(self):
        messagebox.showinfo("Información", "Este es un mensaje")

Dialogo()
```

Ejercicio 18
Título: Mover etiqueta con botones
Explicación: Usa botones para mover una etiqueta dentro de la ventana.
Solución:

```
import tkinter as tk

class MoverEtiqueta:
    def __init__(self):
        self.ventana = tk.Tk()
        self.etiqueta = tk.Label(self.ventana, text="Moverme")
        self.etiqueta.place(x=50, y=50)
```

```
        tk.Button(self.ventana, text="Izquierda", command=lambda:
self.mover(-10, 0)).pack(side='left')
        tk.Button(self.ventana, text="Derecha", command=lambda:
self.mover(10, 0)).pack(side='left')
        tk.Button(self.ventana, text="Arriba", command=lambda:
self.mover(0, -10)).pack(side='left')
        tk.Button(self.ventana, text="Abajo", command=lambda:
self.mover(0, 10)).pack(side='left')
        self.ventana.mainloop()

    def mover(self, dx, dy):
        x = self.etiqueta.winfo_x() + dx
        y = self.etiqueta.winfo_y() + dy
        self.etiqueta.place(x=x, y=y)

MoverEtiqueta()
```

Ejercicio 19
Título: Crear y eliminar etiquetas dinámicamente
Explicación: Botón crea etiquetas y otro las elimina todas.
Solución:

```
import tkinter as tk

class GestionEtiquetas:
    def __init__(self):
        self.ventana = tk.Tk()
        self.etiquetas = []
        tk.Button(self.ventana, text="Crear etiqueta",
command=self.crear).pack()
        tk.Button(self.ventana, text="Eliminar etiquetas",
command=self.eliminar).pack()
        self.ventana.mainloop()

    def crear(self):
        etiqueta = tk.Label(self.ventana, text="Etiqueta")
```

```
        etiqueta.pack()
        self.etiquetas.append(etiqueta)

    def eliminar(self):
        for etiqueta in self.etiquetas:
            etiqueta.destroy()
        self.etiquetas.clear()

GestionEtiquetas()
```

Ejercicio 20
Título: Aplicación calculadora con historial
Explicación: Calculadora que guarda las operaciones realizadas en una lista visible.
Solución:

```
import tkinter as tk

class CalculadoraHistorial:
    def __init__(self):
        self.ventana = tk.Tk()
        self.e1 = tk.Entry(self.ventana)
        self.e2 = tk.Entry(self.ventana)
        self.e1.pack()
        self.e2.pack()
        tk.Button(self.ventana, text="Sumar",
command=self.sumar).pack()
        self.resultado = tk.Label(self.ventana, text="")
        self.resultado.pack()
        self.historial = tk.Listbox(self.ventana)
        self.historial.pack()
        self.ventana.mainloop()

    def sumar(self):
        try:
```

```
            n1 = float(self.e1.get())
            n2 = float(self.e2.get())
            r = n1 + n2
            self.resultado.config(text="Resultado: " + str(r))
            self.historial.insert(tk.END, f"{n1} + {n2} = {r}")
        except:
            self.resultado.config(text="Entrada inválida")

CalculadoraHistorial()
```

---

10. TRATAMIENTO DE FICHEROS CSV

Formato y estructura

Es un formato de texto plano, muy simple.

Cada línea representa un registro o fila.

Los campos o columnas están separados por comas (o a veces otros separadores como punto y coma).

No tiene estructura jerárquica ni soporte para datos anidados.

Ideal para datos tabulares simples, como tablas con filas y columnas.

No puede manejar bien datos jerárquicos o estructurados (por ejemplo, listas dentro de campos).

---

1. Abrir y mostrar contenido CSV en Text widget

```
import tkinter as tk
from tkinter import filedialog
import csv

def abrir_csv():
    ruta = filedialog.askopenfilename(filetypes=[("CSV files",
"*.csv")])
    if ruta:
        with open(ruta, newline='') as archivo:
            lector = csv.reader(archivo)
            texto.delete("1.0", tk.END)
            for fila in lector:
                texto.insert(tk.END, ", ".join(fila) + "\n")

root = tk.Tk()
root.title("Abrir CSV")

btn_abrir = tk.Button(root, text="Abrir CSV", command=abrir_csv)
btn_abrir.pack()

texto = tk.Text(root, width=80, height=20)
texto.pack()

root.mainloop()
```

Explicación: Este programa abre un diálogo para seleccionar un archivo CSV y luego muestra su contenido línea por línea en un widget Text.

---

2. Guardar contenido del Text widget en CSV

```
import tkinter as tk
from tkinter import filedialog
import csv

def guardar_csv():
    ruta = filedialog.asksaveasfilename(defaultextension=".csv",
filetypes=[("CSV files", "*.csv")])
    if ruta:
        contenido = texto.get("1.0", tk.END).strip().split("\n")
        with open(ruta, 'w', newline='') as archivo:
            escritor = csv.writer(archivo)
            for linea in contenido:
                escritor.writerow(linea.split(","))

root = tk.Tk()
root.title("Guardar CSV")

texto = tk.Text(root, width=80, height=20)
texto.pack()

btn_guardar = tk.Button(root, text="Guardar CSV",
command=guardar_csv)
btn_guardar.pack()

root.mainloop()
```

Explicación: Permite escribir en un Text widget, y guardar su contenido como CSV, separando columnas por comas.

---

3. Añadir fila a CSV existente

```
import tkinter as tk
from tkinter import filedialog, simpledialog
import csv

def añadir_fila():
    ruta = filedialog.askopenfilename(filetypes=[("CSV files",
"*.csv")])
    if ruta:
        fila = simpledialog.askstring("Añadir fila", "Introduce
los valores separados por coma:")
        if fila:
            with open(ruta, 'a', newline='') as archivo:
                escritor = csv.writer(archivo)
                escritor.writerow(fila.split(","))

root = tk.Tk()
root.title("Añadir fila a CSV")

btn_añadir = tk.Button(root, text="Añadir fila",
command=añadir_fila)
```

```
btn_añadir.pack()

root.mainloop()
```

Explicación: Abre un CSV y añade una fila al final, que se pide al usuario como texto separado por comas.

---

4. Buscar filas que contengan un texto

```
import tkinter as tk
from tkinter import filedialog, simpledialog
import csv

def buscar():
    ruta = filedialog.askopenfilename(filetypes=[("CSV files",
"*.csv")])
    if ruta:
        termino = simpledialog.askstring("Buscar", "Introduce
texto a buscar:")
        if termino:
            resultado.delete("1.0", tk.END)
            with open(ruta, newline='') as archivo:
                lector = csv.reader(archivo)
                for fila in lector:
                    if any(termino in celda for celda in fila):
                        resultado.insert(tk.END, ", ".join(fila) +
"\n")

root = tk.Tk()
root.title("Buscar en CSV")

btn_buscar = tk.Button(root, text="Buscar", command=buscar)
btn_buscar.pack()

resultado = tk.Text(root, width=80, height=20)
resultado.pack()

root.mainloop()
```

Explicación: Permite buscar un término en todas las filas del CSV y muestra las filas donde aparece.

---

5. Contar filas de un CSV

```
import tkinter as tk
from tkinter import filedialog, messagebox
import csv

def contar_filas():
```

```
    ruta = filedialog.askopenfilename(filetypes=[("CSV files",
"*.csv")])
    if ruta:
        with open(ruta, newline='') as archivo:
            lector = csv.reader(archivo)
            filas = sum(1 for _ in lector)
        messagebox.showinfo("Número de filas", f"El archivo tiene
{filas} filas.")

root = tk.Tk()
root.title("Contar filas CSV")

btn_contar = tk.Button(root, text="Contar filas",
command=contar_filas)
btn_contar.pack()

root.mainloop()
```

Explicación: Muestra en un mensaje cuántas filas tiene el archivo CSV.

---

6. Mostrar solo una columna específica

```
import tkinter as tk
from tkinter import filedialog, simpledialog
import csv

def mostrar_columna():
    ruta = filedialog.askopenfilename(filetypes=[("CSV files",
"*.csv")])
    if ruta:
        col = simpledialog.askinteger("Columna", "Número de
columna (empezando en 0):")
        if col is not None:
            texto_columna.delete("1.0", tk.END)
            with open(ruta, newline='') as archivo:
                lector = csv.reader(archivo)
                for fila in lector:
                    if len(fila) > col:
                        texto_columna.insert(tk.END, fila[col] +
"\n")

root = tk.Tk()
root.title("Mostrar columna CSV")

btn_mostrar = tk.Button(root, text="Mostrar columna",
command=mostrar_columna)
btn_mostrar.pack()

texto_columna = tk.Text(root, width=40, height=20)
texto_columna.pack()

root.mainloop()
```

Explicación: Permite mostrar solo los datos de una columna indicada por el usuario.

---

7. Convertir CSV a mayúsculas

```
import tkinter as tk
from tkinter import filedialog
import csv

def convertir_mayusculas():
    ruta = filedialog.askopenfilename(filetypes=[("CSV files",
"*.csv")])
    if ruta:
        contenido_mayus = []
        with open(ruta, newline='') as archivo:
            lector = csv.reader(archivo)
            for fila in lector:
                contenido_mayus.append([celda.upper() for celda in
fila])
        texto.delete("1.0", tk.END)
        for fila in contenido_mayus:
            texto.insert(tk.END, ", ".join(fila) + "\n")

root = tk.Tk()
root.title("CSV a Mayúsculas")

btn_convertir = tk.Button(root, text="Convertir a Mayúsculas",
command=convertir_mayusculas)
btn_convertir.pack()

texto = tk.Text(root, width=80, height=20)
texto.pack()

root.mainloop()
```

Explicación: Lee el CSV y muestra su contenido convertido a mayúsculas.

---

8. Eliminar una fila por número

```
import tkinter as tk
from tkinter import filedialog, simpledialog, messagebox
import csv

def eliminar_fila():
    ruta = filedialog.askopenfilename(filetypes=[("CSV files",
"*.csv")])
    if ruta:
        num_fila = simpledialog.askinteger("Eliminar fila",
"Número de fila a eliminar (empezando en 0):")
        if num_fila is not None:
            with open(ruta, newline='') as archivo:
```

```
                filas = list(csv.reader(archivo))
            if 0 <= num_fila < len(filas):
                filas.pop(num_fila)
                with open(ruta, 'w', newline='') as archivo:
                    escritor = csv.writer(archivo)
                    escritor.writerows(filas)
                messagebox.showinfo("Éxito", "Fila eliminada
correctamente.")
            else:
                messagebox.showerror("Error", "Número de fila
fuera de rango.")

root = tk.Tk()
root.title("Eliminar fila CSV")

btn_eliminar = tk.Button(root, text="Eliminar fila",
command=eliminar_fila)
btn_eliminar.pack()

root.mainloop()
```

Explicación: Elimina la fila especificada del CSV y guarda los cambios.

---

9. Contar ocurrencias de un valor en una columna

```
import tkinter as tk
from tkinter import filedialog, simpledialog, messagebox
import csv

def contar_valor():
    ruta = filedialog.askopenfilename(filetypes=[("CSV files",
"*.csv")])
    if ruta:
        col = simpledialog.askinteger("Columna", "Número de
columna (empezando en 0):")
        valor = simpledialog.askstring("Valor", "Valor a buscar:")
        if col is not None and valor is not None:
            contador = 0
            with open(ruta, newline='') as archivo:
                lector = csv.reader(archivo)
                for fila in lector:
                    if len(fila) > col and fila[col] == valor:
                        contador += 1
            messagebox.showinfo("Resultado", f"El valor '{valor}'
aparece {contador} veces en la columna {col}.")

root = tk.Tk()
root.title("Contar valor en CSV")

btn_contar = tk.Button(root, text="Contar valor",
command=contar_valor)
btn_contar.pack()
```

```
root.mainloop()
```

Explicación: Cuenta cuántas veces aparece un valor en una columna dada.

---

10. Reemplazar un valor por otro en todo el CSV

```
import tkinter as tk
from tkinter import filedialog, simpledialog, messagebox
import csv

def reemplazar_valor():
    ruta = filedialog.askopenfilename(filetypes=[("CSV files",
"*.csv")])
    if ruta:
        valor_viejo = simpledialog.askstring("Valor viejo", "Valor
a reemplazar:")
        valor_nuevo = simpledialog.askstring("Valor nuevo", "Nuevo
valor:")
        if valor_viejo is not None and valor_nuevo is not None:
            with open(ruta, newline='') as archivo:
                filas = list(csv.reader(archivo))
            for i in range(len(filas)):
                filas[i] = [valor_nuevo if celda == valor_viejo
else celda for celda in filas[i]]
            with open(ruta, 'w', newline='') as archivo:
                escritor = csv.writer(archivo)
                escritor.writerows(filas)
            messagebox.showinfo("Éxito", f"Se reemplazó
'{valor_viejo}' por '{valor_nuevo}' en todo el archivo.")

root = tk.Tk()
root.title("Reemplazar valor en CSV")

btn_reemplazar = tk.Button(root, text="Reemplazar valor",
command=reemplazar_valor)
btn_reemplazar.pack()

root.mainloop()
```

Explicación: Recorre todo el CSV y reemplaza todas las ocurrencias de un valor por otro, guardando el archivo.

---

11. Ordenar CSV por una columna

```
import tkinter as tk

from tkinter import filedialog, simpledialog, messagebox

import csv
```

```
def ordenar_csv():
    ruta = filedialog.askopenfilename(filetypes=[("CSV files", "*.csv")])
    if ruta:
        col = simpledialog.askinteger("Columna", "Número de columna para ordenar (empezando en 0):")
        if col is not None:
            with open(ruta, newline='') as archivo:
                filas = list(csv.reader(archivo))
            try:
                filas_ordenadas = sorted(filas, key=lambda x: x[col])
                texto.delete("1.0", tk.END)
                for fila in filas_ordenadas:
                    texto.insert(tk.END, ", ".join(fila) + "\n")
            except IndexError:
                messagebox.showerror("Error", "Número de columna fuera de rango.")

root = tk.Tk()
root.title("Ordenar CSV")

btn_ordenar = tk.Button(root, text="Ordenar CSV", command=ordenar_csv)
btn_ordenar.pack()

texto = tk.Text(root, width=80, height=20)
texto.pack()

root.mainloop()
```

Explicación: Ordena las filas del CSV por la columna indicada y muestra el resultado en pantalla.

---

12. Exportar CSV filtrando filas por condición

```
import tkinter as tk
from tkinter import filedialog, simpledialog, messagebox
import csv

def exportar_filtrado():
    ruta = filedialog.askopenfilename(filetypes=[("CSV files", "*.csv")])
    if ruta:
```

```
        col = simpledialog.askinteger("Columna", "Número de columna para filtrar (empezando en 0):")
        valor = simpledialog.askstring("Valor", "Valor que debe tener la columna para filtrar:")
        if col is not None and valor is not None:
            ruta_guardar = filedialog.asksaveasfilename(defaultextension=".csv", filetypes=[("CSV files", "*.csv")])
            if ruta_guardar:
                with open(ruta, newline='') as archivo:
                    lector = csv.reader(archivo)
                    filas_filtradas = [fila for fila in lector if len(fila) > col and fila[col] == valor]
                with open(ruta_guardar, 'w', newline='') as archivo_salida:
                    escritor = csv.writer(archivo_salida)
                    escritor.writerows(filas_filtradas)
                messagebox.showinfo("Éxito", "Archivo CSV filtrado guardado correctamente.")


root = tk.Tk()
root.title("Exportar CSV filtrado")


btn_filtrar = tk.Button(root, text="Exportar filtrado", command=exportar_filtrado)
btn_filtrar.pack()


root.mainloop()
```

Explicación: Filtra las filas que tienen un valor concreto en una columna y guarda esas filas en un nuevo CSV.

---

13. Mostrar estadísticas básicas de una columna numérica

```
import tkinter as tk
from tkinter import filedialog, simpledialog, messagebox
import csv
import statistics


def estadisticas():
    ruta = filedialog.askopenfilename(filetypes=[("CSV files", "*.csv")])
    if ruta:
```

```
        col = simpledialog.askinteger("Columna", "Número de columna numérica
(empezando en 0):")
        if col is not None:
            numeros = []
            with open(ruta, newline='') as archivo:
                lector = csv.reader(archivo)
                for fila in lector:
                    try:
                        numeros.append(float(fila[col]))
                    except (ValueError, IndexError):
                        pass
            if numeros:
                media = statistics.mean(numeros)
                mediana = statistics.median(numeros)
                desviacion = statistics.stdev(numeros) if len(numeros) > 1 else
0
                mensaje = f"Media: {media:.2f}\nMediana: {mediana:.2f}
\nDesviación estándar: {desviacion:.2f}"
                messagebox.showinfo("Estadísticas", mensaje)
            else:
                messagebox.showerror("Error", "No se encontraron datos numéricos
en esa columna.")


root = tk.Tk()
root.title("Estadísticas CSV")


btn_stats = tk.Button(root, text="Mostrar estadísticas", command=estadisticas)
btn_stats.pack()


root.mainloop()
```

Explicación: Calcula media, mediana y desviación estándar de una columna numérica del CSV.

---

14. Dividir un CSV en varios archivos por valor de columna

```
import tkinter as tk
from tkinter import filedialog, simpledialog, messagebox
import csv
import os
```

```
def dividir_csv():
    ruta = filedialog.askopenfilename(filetypes=[("CSV files", "*.csv")])
    if ruta:
        col = simpledialog.askinteger("Columna", "Número de columna para dividir (empezando en 0):")
        if col is not None:
            carpeta = filedialog.askdirectory()
            if carpeta:
                grupos = {}
                with open(ruta, newline='') as archivo:
                    lector = csv.reader(archivo)
                    for fila in lector:
                        if len(fila) > col:
                            clave = fila[col]
                            grupos.setdefault(clave, []).append(fila)
                for clave, filas in grupos.items():
                    nombre_archivo = os.path.join(carpeta, f"{clave}.csv")
                    with open(nombre_archivo, 'w', newline='') as archivo_salida:
                        escritor = csv.writer(archivo_salida)
                        escritor.writerows(filas)
                messagebox.showinfo("Éxito", "CSV dividido correctamente por valores de la columna.")

root = tk.Tk()
root.title("Dividir CSV")

btn_dividir = tk.Button(root, text="Dividir CSV", command=dividir_csv)
btn_dividir.pack()

root.mainloop()
```

Explicación: Crea varios CSV separados según el valor de una columna, guardándolos en una carpeta elegida.

---

15. Añadir columna con valores calculados

```
import tkinter as tk
from tkinter import filedialog, simpledialog, messagebox
```

```
import csv


def añadir_columna():
    ruta = filedialog.askopenfilename(filetypes=[("CSV files", "*.csv")])
    if ruta:
        valor = simpledialog.askstring("Valor", "Valor fijo para nueva
columna:")
        if valor is not None:
            with open(ruta, newline='') as archivo:
                filas = list(csv.reader(archivo))
            for fila in filas:
                fila.append(valor)
            ruta_guardar = filedialog.asksaveasfilename(defaultextension=".csv",
filetypes=[("CSV files", "*.csv")])
            if ruta_guardar:
                with open(ruta_guardar, 'w', newline='') as archivo_salida:
                    escritor = csv.writer(archivo_salida)
                    escritor.writerows(filas)
                messagebox.showinfo("Éxito", "Columna añadida y archivo
guardado.")


root = tk.Tk()
root.title("Añadir columna CSV")


btn_añadir = tk.Button(root, text="Añadir columna", command=añadir_columna)
btn_añadir.pack()


root.mainloop()
```

Explicación: Añade una nueva columna con un valor fijo en todas las filas y guarda el CSV modificado.

---

16. Contar ocurrencias de valores en una columna

```
import tkinter as tk
from tkinter import filedialog, simpledialog, messagebox
import csv
from collections import Counter

def contar_ocurrencias():
```

```
    ruta = filedialog.askopenfilename(filetypes=[("CSV files",
"*.csv")])
    if ruta:
        col = simpledialog.askinteger("Columna", "Número de
columna para contar ocurrencias (empezando en 0):")
        if col is not None:
            valores = []
            with open(ruta, newline='') as archivo:
                lector = csv.reader(archivo)
                for fila in lector:
                    if len(fila) > col:
                        valores.append(fila[col])
            contador = Counter(valores)
            resultado = "\n".join(f"{clave}: {valor}" for clave,
valor in contador.items())
            messagebox.showinfo("Ocurrencias", resultado)

root = tk.Tk()
root.title("Contar ocurrencias CSV")

btn_contar = tk.Button(root, text="Contar ocurrencias",
command=contar_ocurrencias)
btn_contar.pack()

root.mainloop()
```

Explicación: Cuenta cuántas veces aparece cada valor en la columna indicada y muestra el conteo.

---

17. Mostrar contenido CSV en tabla con Tkinter (Treeview)

```
import tkinter as tk
from tkinter import filedialog, ttk
import csv

def mostrar_csv():
    ruta = filedialog.askopenfilename(filetypes=[("CSV files",
"*.csv")])
    if ruta:
        for i in tree.get_children():
            tree.delete(i)
        with open(ruta, newline='') as archivo:
            lector = csv.reader(archivo)
            filas = list(lector)
            if filas:
                tree["columns"] = [f"col{i}" for i in
range(len(filas[0]))]
                tree["show"] = "headings"
                for i, col in enumerate(tree["columns"]):
                    tree.heading(col, text=f"Col {i}")
                for fila in filas:
                    tree.insert("", "end", values=fila)
```

```
root = tk.Tk()
root.title("Mostrar CSV en tabla")

btn_mostrar = tk.Button(root, text="Mostrar CSV",
command=mostrar_csv)
btn_mostrar.pack()

tree = ttk.Treeview(root)
tree.pack(expand=True, fill='both')

root.mainloop()
```

Explicación: Lee un CSV y muestra su contenido en una tabla dentro de la ventana usando Treeview.

---

18. Actualizar valor en fila y columna específica

```
import tkinter as tk
from tkinter import filedialog, simpledialog, messagebox
import csv

def actualizar_valor():
    ruta = filedialog.askopenfilename(filetypes=[("CSV files",
"*.csv")])
    if ruta:
        fila_num = simpledialog.askinteger("Fila", "Número de fila
a modificar (empezando en 0):")
        col_num = simpledialog.askinteger("Columna", "Número de
columna a modificar (empezando en 0):")
        nuevo_valor = simpledialog.askstring("Nuevo valor", "Nuevo
valor para la celda:")
        if fila_num is not None and col_num is not None and
nuevo_valor is not None:
            with open(ruta, newline='') as archivo:
                filas = list(csv.reader(archivo))
            try:
                filas[fila_num][col_num] = nuevo_valor
                ruta_guardar =
filedialog.asksaveasfilename(defaultextension=".csv",
filetypes=[("CSV files", "*.csv")])
                if ruta_guardar:
                    with open(ruta_guardar, 'w', newline='') as
archivo_salida:
                        escritor = csv.writer(archivo_salida)
                        escritor.writerows(filas)
                    messagebox.showinfo("Éxito", "Archivo
actualizado y guardado.")
            except IndexError:
                messagebox.showerror("Error", "Fila o columna
fuera de rango.")
```

```
root = tk.Tk()
root.title("Actualizar CSV")

btn_actualizar = tk.Button(root, text="Actualizar valor",
command=actualizar_valor)
btn_actualizar.pack()

root.mainloop()
```

Explicación: Permite modificar un valor concreto en el CSV y guardar el archivo actualizado.

---

19. Eliminar filas con columna vacía

```
import tkinter as tk
from tkinter import filedialog, simpledialog, messagebox
import csv

def eliminar_filas_vacias():
    ruta = filedialog.askopenfilename(filetypes=[("CSV files",
"*.csv")])
    if ruta:
        col = simpledialog.askinteger("Columna", "Número de
columna a comprobar (empezando en 0):")
        if col is not None:
            with open(ruta, newline='') as archivo:
                lector = csv.reader(archivo)
                filas = [fila for fila in lector if len(fila) >
col and fila[col].strip() != ""]
            ruta_guardar =
filedialog.asksaveasfilename(defaultextension=".csv",
filetypes=[("CSV files", "*.csv")])
            if ruta_guardar:
                with open(ruta_guardar, 'w', newline='') as
archivo_salida:
                    escritor = csv.writer(archivo_salida)
                    escritor.writerows(filas)
                messagebox.showinfo("Éxito", "Filas con columna
vacía eliminadas y archivo guardado.")

root = tk.Tk()
root.title("Eliminar filas vacías")

btn_eliminar = tk.Button(root, text="Eliminar filas vacías",
command=eliminar_filas_vacias)
btn_eliminar.pack()

root.mainloop()
```

Explicación: Elimina las filas donde la columna indicada está vacía y guarda el CSV resultante.

---

20. Combinar dos CSV y mostrar resultado

```
import tkinter as tk
from tkinter import filedialog, messagebox
import csv

def combinar_csv():
    ruta1 = filedialog.askopenfilename(title="Seleccionar primer
CSV", filetypes=[("CSV files", "*.csv")])
    if not ruta1:
        return
    ruta2 = filedialog.askopenfilename(title="Seleccionar segundo
CSV", filetypes=[("CSV files", "*.csv")])
    if not ruta2:
        return
    filas_combinadas = []
    with open(ruta1, newline='') as archivo1:
        lector1 = csv.reader(archivo1)
        filas_combinadas.extend(list(lector1))
    with open(ruta2, newline='') as archivo2:
        lector2 = csv.reader(archivo2)
        filas_combinadas.extend(list(lector2))
    texto.delete("1.0", tk.END)
    for fila in filas_combinadas:
        texto.insert(tk.END, ", ".join(fila) + "\n")
    messagebox.showinfo("Éxito", "CSV combinados y mostrados.")

root = tk.Tk()
root.title("Combinar CSV")

btn_combinar = tk.Button(root, text="Combinar CSV",
command=combinar_csv)
btn_combinar.pack()

texto = tk.Text(root, width=80, height=20)
texto.pack()

root.mainloop()
```

Explicación: Combina dos archivos CSV uno tras otro y muestra el resultado en pantalla.

---

11. TRATAMIENTO DE FICHEROS JSON

Es un formato de texto estructurado basado en objetos y arrays (listas).

Puede representar datos complejos y jerárquicos, con anidamientos, objetos dentro de objetos, listas dentro de objetos, etc.

Es muy usado para intercambio de datos en aplicaciones web.

Puede manejar estructuras complejas, objetos, listas y anidamientos.

Es muy flexible para representar cualquier tipo de dato.

Ejercicio 1
Enunciado: Crear una interfaz para guardar un diccionario simple en un fichero JSON.
Solución:

```
import tkinter as tk
import json

def guardar_json():
    datos = {"nombre": entrada.get()}
    with open("datos1.json", "w") as f:
        json.dump(datos, f)

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Guardar", command=guardar_json).pack()
ventana.mainloop()
```

Explicación: Se toma el valor del Entry y se guarda como JSON.

Ejercicio 2
Enunciado: Leer un archivo JSON y mostrar su contenido en un Text.
Solución:

```
import tkinter as tk
import json

def leer_json():
```

```
    with open("datos1.json", "r") as f:
        datos = json.load(f)
    texto.insert(tk.END, str(datos))

ventana = tk.Tk()
texto = tk.Text(ventana)
texto.pack()
tk.Button(ventana, text="Leer", command=leer_json).pack()
ventana.mainloop()
```

Explicación: Se carga el archivo y se muestra en el widget Text.

Ejercicio 3
Enunciado: Crear una GUI para agregar múltiples personas a un JSON.
Solución:

```
import tkinter as tk
import json
import os

def agregar_persona():
    nombre = entrada.get()
    if os.path.exists("personas.json"):
        with open("personas.json", "r") as f:
            datos = json.load(f)
    else:
        datos = []
    datos.append({"nombre": nombre})
    with open("personas.json", "w") as f:
        json.dump(datos, f)

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Agregar", command=agregar_persona).pack()
ventana.mainloop()
```

Explicación: Se agregan objetos al array JSON sin borrar los anteriores.

Ejercicio 4
Enunciado: Mostrar nombres almacenados en el JSON en una lista.
Solución:

```
import tkinter as tk
import json

def cargar():
    with open("personas.json", "r") as f:
        datos = json.load(f)
    for persona in datos:
        lista.insert(tk.END, persona["nombre"])

ventana = tk.Tk()
lista = tk.Listbox(ventana)
lista.pack()
tk.Button(ventana, text="Cargar", command=cargar).pack()
ventana.mainloop()
```

Explicación: Los nombres del JSON se insertan en la Listbox.

Ejercicio 5
Enunciado: Buscar una persona por nombre en el JSON.
Solución:

```
import tkinter as tk
import json

def buscar():
    nombre = entrada.get()
    with open("personas.json", "r") as f:
        datos = json.load(f)
    for persona in datos:
        if persona["nombre"] == nombre:
            resultado.set("Encontrado")
            return
    resultado.set("No encontrado")

ventana = tk.Tk()
entrada = tk.Entry(ventana)
```

```
entrada.pack()
resultado = tk.StringVar()
tk.Label(ventana, textvariable=resultado).pack()
tk.Button(ventana, text="Buscar", command=buscar).pack()
ventana.mainloop()
```

Explicación: Se compara el nombre introducido con los almacenados.

Ejercicio 6
Enunciado: Editar un nombre ya existente en el fichero JSON.
Solución:

```
import tkinter as tk
import json

def editar():
    antiguo = entrada1.get()
    nuevo = entrada2.get()
    with open("personas.json", "r") as f:
        datos = json.load(f)
    for persona in datos:
        if persona["nombre"] == antiguo:
            persona["nombre"] = nuevo
            break
    with open("personas.json", "w") as f:
        json.dump(datos, f)

ventana = tk.Tk()
entrada1 = tk.Entry(ventana)
entrada1.pack()
entrada2 = tk.Entry(ventana)
entrada2.pack()
tk.Button(ventana, text="Editar", command=editar).pack()
ventana.mainloop()
```

Explicación: Se reemplaza el nombre antiguo por el nuevo.

Ejercicio 7
Enunciado: Eliminar un nombre del JSON.
Solución:

```
import tkinter as tk
import json

def eliminar():
    nombre = entrada.get()
    with open("personas.json", "r") as f:
        datos = json.load(f)
    datos = [p for p in datos if p["nombre"] != nombre]
    with open("personas.json", "w") as f:
        json.dump(datos, f)

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Eliminar", command=eliminar).pack()
ventana.mainloop()
```

Explicación: Se filtran los elementos que no coinciden con el nombre.

Ejercicio 8
Enunciado: Guardar y leer múltiples campos (nombre, edad) en JSON.
Solución:

```
import tkinter as tk
import json

def guardar():
    nombre = entry_nombre.get()
    edad = entry_edad.get()
    persona = {"nombre": nombre, "edad": edad}
    with open("datos_completo.json", "w") as f:
        json.dump(persona, f)

ventana = tk.Tk()
entry_nombre = tk.Entry(ventana)
entry_nombre.pack()
entry_edad = tk.Entry(ventana)
entry_edad.pack()
tk.Button(ventana, text="Guardar", command=guardar).pack()
```

```
ventana.mainloop()
```

Explicación: Se guarda un diccionario con múltiples campos.

Ejercicio 9
Enunciado: Leer múltiples campos de un JSON y mostrarlos.
Solución:

```
import tkinter as tk
import json

def leer():
    with open("datos_completo.json", "r") as f:
        datos = json.load(f)
    texto.set(f"{datos['nombre']} - {datos['edad']} años")

ventana = tk.Tk()
texto = tk.StringVar()
tk.Label(ventana, textvariable=texto).pack()
tk.Button(ventana, text="Leer", command=leer).pack()
ventana.mainloop()
```

Explicación: Se muestran los datos del JSON usando StringVar.

Ejercicio 10
Enunciado: Crear archivo JSON vacío desde Tkinter.
Solución:

```
import tkinter as tk
import json

def crear_vacio():
    with open("vacio.json", "w") as f:
        json.dump([], f)

ventana = tk.Tk()
tk.Button(ventana, text="Crear JSON vacío", command=crear_vacio).pack()
ventana.mainloop()
```

Explicación: Se inicializa el archivo con una lista vacía.

Ejercicio 11
Enunciado: Añadir elementos a un array JSON desde la interfaz.
Solución:

```
import tkinter as tk
import json
import os

def añadir():
    nuevo = entrada.get()
    if os.path.exists("array.json"):
        with open("array.json", "r") as f:
            datos = json.load(f)
    else:
        datos = []
    datos.append(nuevo)
    with open("array.json", "w") as f:
        json.dump(datos, f)

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Añadir", command=añadir).pack()
ventana.mainloop()
```

Explicación: Se añade texto ingresado a una lista JSON existente o nueva.

Ejercicio 12
Enunciado: Mostrar el contenido de un JSON de lista línea por línea.
Solución:

```
import tkinter as tk
import json

def mostrar():
    with open("array.json", "r") as f:
        datos = json.load(f)
    lista.delete(0, tk.END)
    for item in datos:
        lista.insert(tk.END, item)
```

```
ventana = tk.Tk()
lista = tk.Listbox(ventana)
lista.pack()
tk.Button(ventana, text="Mostrar", command=mostrar).pack()
ventana.mainloop()
```

Explicación: Cada elemento de la lista JSON se muestra en una línea de Listbox.

Ejercicio 13
Enunciado: Eliminar todo el contenido de un JSON desde la interfaz.
Solución:

```
import tkinter as tk
import json

def limpiar():
    with open("array.json", "w") as f:
        json.dump([], f)

ventana = tk.Tk()
tk.Button(ventana, text="Vaciar JSON", command=limpiar).pack()
ventana.mainloop()
```

Explicación: Se sobrescribe el archivo con una lista vacía.

Ejercicio 14
Enunciado: Guardar configuración de usuario en JSON.
Solución:

```
import tkinter as tk
import json

def guardar_config():
    config = {"tema": tema.get(), "fuente": fuente.get()}
    with open("config.json", "w") as f:
        json.dump(config, f)

ventana = tk.Tk()
tema = tk.Entry(ventana)
tema.pack()
```

```
fuente = tk.Entry(ventana)
fuente.pack()
tk.Button(ventana, text="Guardar Configuración", command=guardar_config).pack()
ventana.mainloop()
```

Explicación: Se almacenan configuraciones en un JSON como diccionario.

Ejercicio 15
Enunciado: Cargar configuración de usuario desde JSON.
Solución:

```
import tkinter as tk
import json

def cargar_config():
    with open("config.json", "r") as f:
        datos = json.load(f)
    salida.set(f"Tema: {datos['tema']}, Fuente: {datos['fuente']}")

ventana = tk.Tk()
salida = tk.StringVar()
tk.Label(ventana, textvariable=salida).pack()
tk.Button(ventana, text="Cargar Configuración", command=cargar_config).pack()
ventana.mainloop()
```

Explicación: Se lee el archivo de configuración y se muestra al usuario.

Ejercicio 16
Enunciado: Crear interfaz para exportar datos a JSON desde múltiples campos.
Solución:

```
import tkinter as tk
import json

def exportar():
    datos = {
        "nombre": nombre.get(),
        "edad": edad.get(),
        "ciudad": ciudad.get()
    }
    with open("exportado.json", "w") as f:
```

```
        json.dump(datos, f)

ventana = tk.Tk()
nombre = tk.Entry(ventana)
nombre.pack()
edad = tk.Entry(ventana)
edad.pack()
ciudad = tk.Entry(ventana)
ciudad.pack()
tk.Button(ventana, text="Exportar", command=exportar).pack()
ventana.mainloop()
```

Explicación: Se toman datos de tres Entry y se exportan a JSON.

Ejercicio 17
Enunciado: Verificar si un archivo JSON existe desde la interfaz.
Solución:

```
import tkinter as tk
import os

def verificar():
    if os.path.exists("datos1.json"):
        resultado.set("Existe")
    else:
        resultado.set("No existe")

ventana = tk.Tk()
resultado = tk.StringVar()
tk.Label(ventana, textvariable=resultado).pack()
tk.Button(ventana, text="Verificar Archivo", command=verificar).pack()
ventana.mainloop()
```

Explicación: Se verifica la existencia del archivo antes de abrirlo.

Ejercicio 18
Enunciado: Crear un historial de cambios en JSON.
Solución:

```
import tkinter as tk
import json
```

```
import os

def registrar():
    entrada_texto = entrada.get()
    if os.path.exists("historial.json"):
        with open("historial.json", "r") as f:
            historial = json.load(f)
    else:
        historial = []
    historial.append(entrada_texto)
    with open("historial.json", "w") as f:
        json.dump(historial, f)

ventana = tk.Tk()
entrada = tk.Entry(ventana)
entrada.pack()
tk.Button(ventana, text="Registrar", command=registrar).pack()
ventana.mainloop()
```

Explicación: Cada nuevo texto se añade a un historial JSON.

Ejercicio 19
Enunciado: Mostrar el historial desde archivo JSON.
Solución:

```
import tkinter as tk
import json

def ver_historial():
    with open("historial.json", "r") as f:
        datos = json.load(f)
    texto.delete("1.0", tk.END)
    for linea in datos:
        texto.insert(tk.END, linea + "\n")

ventana = tk.Tk()
texto = tk.Text(ventana)
texto.pack()
tk.Button(ventana, text="Ver Historial", command=ver_historial).pack()
```

```
ventana.mainloop()
```

Explicación: Muestra el historial línea por línea desde el JSON.

Ejercicio 20
Enunciado: Copiar un JSON a otro archivo desde la interfaz.
Solución:

```
import tkinter as tk
import json
import shutil

def copiar():
    shutil.copy("datos1.json", "copia_datos.json")

ventana = tk.Tk()
tk.Button(ventana, text="Copiar Archivo JSON", command=copiar).pack()
ventana.mainloop()
```

Explicación: Se usa `shutil.copy` para duplicar el archivo JSON.